



Advanced Class

Paul Barker, Henry Bruce, Robert Berger, Stephano Cetola, Beth Flanagan,
Scott Murray, Khem Raj, David Reyna, Marek Vasut, Trevor Woerner

**Yocto Project Developer Day •
Edinburg • 25 October 2018**

Advanced Class

- **Class Content (download these slides!):**
 - https://wiki.yoctoproject.org/wiki/DevDay_Edinburgh_2018
- **Requirements:**
 - Wireless connection: same as ELCE conference
 - SSH (Windows: e.g. “putty”)
- **Wireless Registration:**
 - Will be passed out

Agenda – The Advanced Class

9:00- 9:15	Keynote
9:15- 9:45	Package Feeds
9:45-10:15	Slim Bootloader
10:15-10:30	Morning Break
10:30-11:15	U-Boot Bootloader
11:15-12:00	Devtool, Next steps
12:00-12:45	Lunch
12:45- 1:45	Licensing 2.0
1:45- 2:15	Device Trees 2.0
2:30- 2:45	Afternoon Break
2:45- 3:15	Yocto Project - Rarely asked questions
3:15- 3:45	(Fun with) Libraries/SDK and OE/YP
3:45- 4:15	Image Size Reduction
4:15- 5:00	Tools, Toaster, User Experience
5:00- 5:30	Forum, Q and A



Class Account Setup

Yocto Project Dev Day Lab Setup

- **The virtual host's resources can be found here:**
 - Your Project: `"/scratch/poky/build-qemuarm"`
 - Extensible-SDK Install: `"/scratch/sdk/qemuarm"`
 - Sources: `"/scratch/src"`
 - Poky: `"/scratch/poky"`
 - Downloads: `"/scratch/downloads"`
 - Sstate-cache: `"/scratch/sstate-cache"`
- **You will be using SSH to communicate with your virtual server.**

FYI: How class project was prepared (1/2)

```
$  
$ cd /scratch  
$ git clone -b sumo git://git.yoctoproject.org/poky.git  
$ cd poky  
$  
$ bash # set up local shell  
$ # Prepare the project  
$ ./scratch/poky/oe-init-build-env build  
$ echo "MACHINE = \"qemuarm\"" >> conf/local.conf  
$ echo "SSTATE_DIR = \"/scratch/sstate-cache\"" >> conf/local.conf  
$ echo "DL_DIR = \"/scratch/downloads\"" >> conf/local.conf  
$ echo "IMAGE_INSTALL_append = \" gdbserver openssh libstdc++ \  
    curl \"" >> conf/local.conf  
$  
$ # Build the project  
$ bitbake core-image-base  
$
```

FYI: How class project was prepared (2/2)

```
$ # Build the eSDK
$
$ bitbake core-image-base -c populate_sdk_ext
$ cd /scratch/poky/build/tmp/deploy/sdk/
$ ./poky-glibc-x86_64-core-image-base-armv5e-toolchain-ext-*.sh \
    -y -d /scratch/sdk/qemuarm
$ exit                                     # return to clean shell
$
$
$ bash                                     # set up local shell
$ cd /scratch/sdk/qemuarm
$ . /scratch/sdk/qemuarm/environment-setup-armv5e-poky-linux-gnueabi
$ devtool modify virtual/kernel
$ exit                                     # return to clean shell
$
```

NOTE: Clean Shells!

- We are going to do a lot of different exercises in different build projects, each with their own environments.
- To keep things sane, you should have a new clean shell for each exercise.
- There are two simple ways to do it:
 1. Close your existing SSH connection and open a new one
-- or --
 2. Do a “bash” before each exercise to get a new sub-shell, and “exit” at the end to remove it, in order to return to a pristine state.



Activity One

Keynote
Nicolas Dechesne

Keynote

- **Slides are available at:**
 - https://wiki.yoctoproject.org/wiki/File:Yocto_Dev_Day_ELCE_18_Intro.pdf
- **Master DevDay Slides Page:**
 - https://wiki.yoctoproject.org/wiki/DevDay_Edinburgh_2018

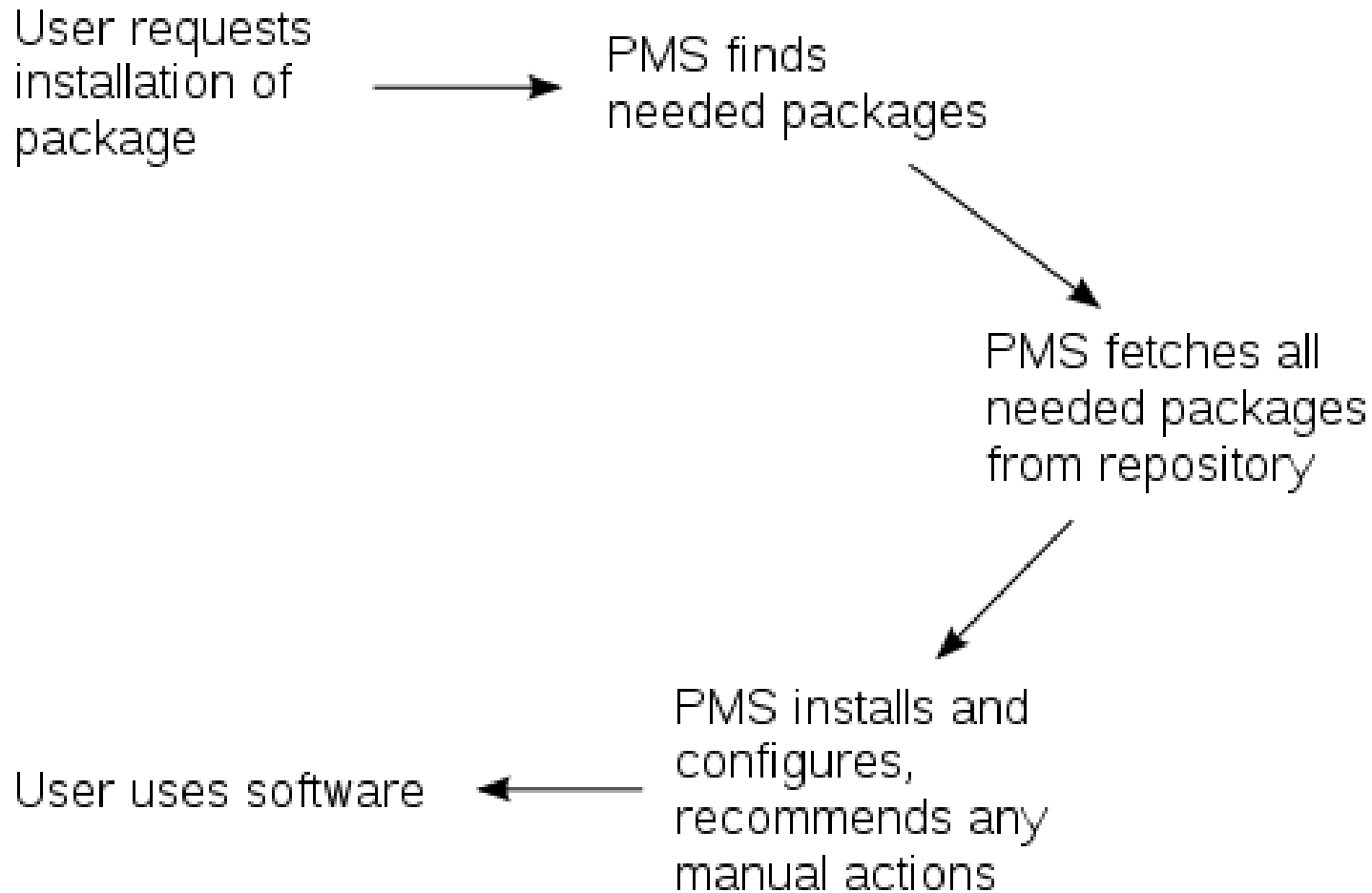


Activity Two

On Target Development using Package Feeds

Stephano Cetola

Package Feed Overview



Package Feed Overview

- Tested package types: rpm and ipk
- For rpm packages, we now use DNF instead of smart
- Setting up a package feed is EASY
 - stephano.cetola@linux.intel.com
 - @stephano approves this message
- Signing your packages and package feed is doable
- Two major use cases:
 - On target development (faster and smarter)
 - In the field updates (YMMV)

On Target Development – Better, Faster, Stronger

Topics

- **Setting up a package feed**
- On target example – AWS + Beaglebone Black
- Signing package feeds
- Keeping your code secure
- The future of package feeds

Setting up a package feed - Target Setup

- **Install Package Management on the target**
 - `EXTRA_IMAGE_FEATURES += " package-management "`
- **Set the correct package class**
 - `PACKAGE_CLASSES = "package_rpm"`
- **Customize the feed (optional)**
 - `PACKAGE_FEED_URI = http://my-server.com/repo`
 - `PACKAGE_FEED_BASE_PATHS = "rpm"`
 - `PACKAGE_FEED_ARCHS = "all armv7at2hf-neon beaglebone"`
- **Edit `/etc/yum.repos.d/oe-remote-repo.repo` (optional)**
 - `enabled=1`
 - `metadata_expire=0`
 - `gpgcheck=0`

Setting up a package feed

- Publish a repo, index the repo, and...

```
$ bitbake core-image-minimal
...
$ bitbake package-index
...
$ twistd -n web --path tmp/deploy/rpm -p 5678
[-] Log opened
[-] twistd 16.0.0 (/usr/bin/python 2.7.12) starting up.
[-] reactor class: twisted.internet.epollreactor.EPollReactor.
[-] Site starting on 5678
```

- You are now running a web server on port 5678

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- **On target example – AWS + Beaglebone Black**
- Signing package feeds
- Keeping your code secure
- The future of package feeds

Caveats

- **Running bitbake world can take some time**
 - You may want to update your repo as needed
- **Serve the repo from a build machine**
 - Or simply rsync to a webserver
- **Do not forget to run `bitbake packge-index`**
 - Package index will not auto-update
- **Good practice is to dogfood your repo**

Understanding RPM Packages and repomd.xml

- **repomd == Repo Metadata**
 - This is the “package index”
- **Repository Tools**
 - createrepo
 - rpm2cpio
 - dnf (replaces yum)
 - yum-utils (historical)
- **Important Commands**
 - rpm -qip (general info)
 - rpm -qpR (depends)
 - <https://wiki.yoctoproject.org/wiki/TipsAndTricks/UsingRPM>



Package Feeds: On Target Demo

Beaglebone Repo on AWS

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- On target example – AWS + Beaglebone Black
- **Signing package feeds**
- Keeping your code secure
- The future of package feeds

Signing The Packages

- **Inherit bbclass to enable signing functionality**
 - `INHERIT += "sign_rpm"`
- **Define the GPG key that will be used for signing.**
 - `RPM_GPG_NAME = "key_name"`
- **Provide passphrase for the key**
 - `RPM_GPG_PASSPHRASE = "passphrase"`

Signing The Package Feed

- **Inherit bbclass to enable signing functionality**
 - `INHERIT += "sign_package_feed"`
- **Define the GPG key that will be used for signing.**
 - `PACKAGE_FEED_GPG_NAME = "key_name"`
- **Provide passphrase for the key**
 - `PACKAGE_FEED_GPG_PASSPHRASE_FILE = "passphrase"`

Signing The Package Feed (optional)

- ***GPG_BIN***
 - GPG binary executed when the package is signed
- ***GPG_PATH***
 - GPG home directory used when the package is signed.
- ***PACKAGE_FEED_GPG_SIGNATURE_TYPE***
 - Specifies the type of gpg signature. This variable applies only to RPM and IPK package feeds. Allowable values for the PACKAGE_FEED_GPG_SIGNATURE_TYPE are "ASC", which is the default and specifies ascii armored, and "BIN", which specifies binary.

Testing Packages with ptest (Optional? Not really!)

- **Package Test (ptest)**

- Runs tests against packages
- Contains at least two items:
 - 1 the actual test (can be a script or an elaborate system)
 - 2 shell script (run-ptest) that starts the test (not the actual test)

- **Simple Setup**

- `DISTRO_FEATURES_append = " ptest"`
- `EXTRA_IMAGE_FEATURES += "ptest-pkgs"`
- Installed to: `/usr/lib/package/ptest`

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- On target example – AWS + Beaglebone Black
- Signing package feeds
- **Keeping your code secure**
- The future of package feeds

Keeping feeds secure

- **PACKAGE_FEED_GPG_PASSPHRASE_FILE**
 - This should NOT go in your configuration as plain text.
- **Is your code proprietary?**
 - You should probably be shipping a binary in Yocto
 - `bin_package.bbclass`: binary can be `.rpm`, `.deb`, `.ipk`
- **Have you thought about DEBUG_FLAGS?**
 - See `bitbake.conf` for more details
 - The flags can be filtered or set in the recipe

On Target Development – Better, Faster, Stronger

Topics

- Setting up a package feed
- On target example – AWS + Beaglebone Black
- Signing package feeds
- Keeping your code secure
- **The future of package feeds**

The Future of Package Feeds – Can We Upgrade?

- Repository
 - Switch to new source entries
 - Remove unknown (3rd party) repositories
- Package
 - Check there are no broken or renamed packages
 - Versioning: what happens when they go backwards
 - Remove and install specific packages (release dependent)
 - Remove blacklisted / obsolete and add whitelisted
- Dreaming Even Bigger...
 - Kernels, Desktops (UI), Permissions, Users, Groups



Activity Three

Slim Bootloader
Stephano Cetola

Slim Bootloader

- **Slides are available at:**
 - https://wiki.yoctoproject.org/wiki/File:Developing_Boot_Solutions_for_Intel_IoT_Unique_Use_Cases_rev1a.pptx
- **Master DevDay Slides Page:**
 - https://wiki.yoctoproject.org/wiki/DevDay_Edinburgh_2018



Activity Four

U-Boot bootloader

Marek Vasut

Booting contemporary hardware

- **Contemporary embedded system boots like this**
 - Power on
 - (optional) BootROM
 - (optional) First stage bootloader
 - Next stage bootloader
 - (optional) other bootloader stages
 - Linux kernel
 - Userspace
- **We will focus on the bootloader parts**

U-Boot bootloader

- **De-facto standard bootloader in embedded**
 - Capable of starting Linux, *BSD, RTOSes, UEFI apps
- **U-Boot is also a boot monitor**
 - U-Boot has a powerful command shell
 - Allows manipulating with the boot process
 - (boot different kernel, script the boot process...)
- **U-Boot is also a debug multitool**
 - U-Boot shell tools allow operating hardware blocks
 - (memory IO, SPI, I2C, network, USB, ...)

Experimenting with U-Boot bootloader

- **Three ways of doing that:**
 - U-Boot sandbox target
 - U-Boot built as a Linux userspace binary
 - QEMU
 - U-Boot running in QEMU
 - Real hardware (danger zone)
 - U-Boot running on real HW
 - Flashing incorrect bootloader brick the device :-)

Experimenting with U-Boot bootloader in OE

- **Use the meta-dto-microdemo layer**
- **Metalayer contains convenience recipes**
 - The u-boot-sandbox recipe
 - To quickly build U-Boot sandbox native target
 - See `recipes-bsp/u-boot/`
 - Kernel config changes to enable virt platform
 - See `recipes-kernel/linux/files/force-virt.cfg`
 - DTO related things for later

DTO Hands-on 1/2

Experimenting with U-Boot bootloader in OE

- **Add meta-dto-demo to bblayers.conf BBLAYERS:**

```
$ echo "BBLAYERS += \"/scratch/src/dto/meta-dto-microdemo\" \" \" \>> conf/bblayers.conf
$ echo "MACHINE = \"qemuarm\" \" \" >> conf/local.conf
$ echo "SSTATE_DIR = \"/scratch/sstate-cache\" \" \" >> conf/local.conf
$ echo "DL_DIR = \"/scratch/downloads\" \" \" >> conf/local.conf
$ echo "UBOOT_MACHINE = \"qemu_arm_defconfig\" \" \" >> conf/local.conf
```

- **Rebuild u-boot, u-boot-sandbox-native and qemu-native**

```
$ bitbake -c cleansstate u-boot u-boot-sandbox-native qemu-native \
    virtual/kernel
$ bitbake u-boot u-boot-sandbox-native u-boot-mkimage-native \
    qemu-native virtual/kernel
```

DTO Hands-on 1/2

Experimenting with U-Boot bootloader in OE

- Start the u-boot sandbox (use Ctrl-C to exit)

```
$ ./tmp/work/x86_64-linux/u-boot-sandbox-native/\
    1_2018.01-r0/git/u-boot
U-Boot 2018.01-dirty (Oct 14 2018 - 11:30:36 +0000)
[...]
SCSI:  Net:  No ethernet found.
IDE:   Bus 0: not available
Hit any key to stop autoboot:  0
=>
=> help
?      - alias for 'help'
base   - print or set address offset
bootz  - boot Linux zImage image from memory
[...]
=> help bootz
bootz - boot Linux zImage image from memory

Usage:
bootz [addr [initrd[:size]] [fdt]]
    - boot Linux zImage stored in memory
    The argument 'initrd' is optional and specifies the address
```

DTO Hands-on 1/2

Experimenting with U-Boot bootloader in OE

- **Start U-Boot in QEMU**

```
$ ./tmp/work/x86_64-linux/qemu-native/\
    2.11.1-r0/build/arm-softmmu/qemu-system-arm \
    -machine virt \
    -bios tmp/deploy/images/qemuarm/u-boot.bin -nographic

U-Boot 2018.01 (Oct 11 2018 - 13:14:21 +0000)

DRAM:  128 MiB
WARNING: Caches not enabled
Using default environment

In:     pl011@90000000
Out:    pl011@90000000
Err:    pl011@90000000
Net:    No ethernet found.
Hit any key to stop autoboot:  0
=>
```

- **(CTRL-A x to quit QEMU)**

DTO Hands-on 1/2

Booting the kernel zImage

- **Generate suitable NOR flash image**

```
$ dd if=/dev/zero of=/tmp/test.bin bs=16M count=0 seek=1
$ dd if=tmp/deploy/images/qemuarm/u-boot.bin \
    of=/tmp/test.bin conv=notrunc
$ dd if=tmp/deploy/images/qemuarm/zImage \
    of=/tmp/test.bin bs=1M seek=1 conv=notrunc
```

- **Start U-Boot with this NOR flash image**

```
$ ./tmp/work/x86_64-linux/qemu-native/2.11.1-r0/\
    build/arm-softmmu/qemu-system-arm \
    -machine virt -bios /tmp/test.bin -nographic
U-Boot 2018.01 (Oct 11 2018 - 13:14:21 +0000)

DRAM: 128 MiB
In: pl011@90000000
Out: pl011@90000000
Err: pl011@90000000
Net: No ethernet found.
Hit any key to stop autoboot: 0
=>
```

DTO Hands-on 1/2

Booting the kernel zImage

- Figure out where the RAM is

```
=> bdi
arch_number = 0x00000000
boot_params = 0x00000000
DRAM bank   = 0x00000000
-> start     = 0x40000000
-> size      = 0x08000000
baudrate     = 115200 bps
TLB addr     = 0x47FF0000
relocaddr    = 0x47F88000
reloc off    = 0x47F88000
irq_sp       = 0x46F66ED0
sp start     = 0x46F66EC0
Early malloc usage: 104 / 400
fdt_blob     = 46f66ee8
```

- Kernel is ~ 8 MiB, copy it to RAM start + 0x8000

```
=> cp 0x100000 0x40008000 0x200000
```

DTO Hands-on 1/2

Booting the kernel zImage

- **Boot the zImage with DT**

```
=> bootz 0x40008000 - $fdtcontroladdr
Kernel image @ 0x40008000 [ 0x000000 - 0x524da0 ]
## Flattened Device Tree blob at 46f66ee8
   Booting using the fdt blob at 0x46f66ee8
   Using Device Tree in place at 46f66ee8, end 46f79ee7

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.14.67-yocto-standard (oe-user@oe-host)
(gcc version 7.3.0 (GCC)) #1 PREEMPT Thu Oct 11 13:10:58 UTC 2018
[    0.000000] CPU: ARMv7 Processor [412fc0f1] revision 1 (ARMv7), cr=10c53c7d
[    0.000000] CPU: div instructions available: patching division code
[    0.000000] CPU: PIPT / VIPT nonaliasing data cache, PIPT instruction cache
[    0.000000] OF: fdt: Machine model: linux,dummy-virt
[    0.000000] Memory policy: Data cache writeback
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv0.2 detected in firmware.
```

- **The \$fdtcontroladdr is DT generated by QEMU**
 - You can dump the DT by appending -machine dumpdtb=file.dtb

Kernel image types – zImage and Image

- **zImage and Image**
 - Linux binary with decompressor
 - No protection against bitrot
 - Set up registers as needed and jump to it
 - DT is optional and separate
 - Boot with U-Boot “bootz” command
 - On Aarch64, similar type of image is called Image
 - Boot with “booti” command
- **ulmage**
- **fitImage**

Kernel image types – **ulmage**

- **zImage and Image**
- **ulmage**
 - Envelope around arbitrary file
 - Legacy since forever
 - Small header with CRC32 and metadata
 - Note that CRC32 is weak
 - Metadata contain payload type, load address...
 - Wraps only one single file
 - Boot with “bootm” command
- **fitImage**

Kernel image types – fitImage

- zImage and Image
- uImage
- FitImage
 - Multi-component image
 - Based on DT
 - Can bundle multiple files with different properties
 - Configurable checksum per-entry
 - CRC32, MD5, SHA1, SHA256...
 - Supports digital signatures
 - RSA2048, RSA4096...

DTO Hands-on 1/2

Building the kernel fitImage

- **QEMU specific step – dump the DTB**

```
=> $ qemu-system-arm -machine virt -nographic -machine dumpdtb=qemu.dtb
```

- **Copy over the fitImage source from the metalayer**

```
$ cp /scratch/src/dto/meta-dto-microdemo/recipes-kernel/\
    linux/files/fit-image.its .
```

- **Build the fitImage:**

```
$ export PATH=$PATH:tmp/work/x86_64-linux/dtc-native/\
    1.4.5-r0/image/scratch/poky/build/tmp/work/x86_64-linux/\
    dtc-native/1.4.5-r0/recipe-sysroot-native/usr/bin/
$ ./tmp/work/x86_64-linux/u-boot-mkimage-native/\
    1_2018.01-r0/git/tools/mkimage -f ./fit-image.its /tmp/fitImage
FIT description: Linux kernel and FDT blob
Created:          Sun Oct 14 12:57:59 2018
Image 0 (kernel-1)
  Description:    Linux kernel
  Created:        Sun Oct 14 12:57:59 2018
  Type:           Kernel Image
  Compression:    uncompressed
  Data Size:      5393824 Bytes = 5267.41 KiB = 5.14 MiB
  Architecture:  ARM
  OS:             Linux
[...]
```

DTO Hands-on 1/2

Booting the kernel fitImage

- **Generate suitable NOR flash image**

```
$ dd if=/dev/zero of=/tmp/test.bin bs=16M count=0 seek=1
$ dd if=tmp/deploy/images/qemuarm/u-boot.bin \
    of=/tmp/test.bin conv=notrunc
$ dd if=/tmp/fitImage \
    of=/tmp/test.bin bs=1M seek=1 conv=notrunc
```

- **Start U-Boot with this NOR flash image**

```
$ ./tmp/work/x86_64-linux/qemu-native/2.11.1-r0/\
    build/arm-softmmu/qemu-system-arm \
    -machine virt -bios /tmp/test.bin -nographic
[...]
=> setenv fdt_high 0x48000000
=> bootm 0x100000
## Loading kernel from FIT Image at 00100000 ...
   Using 'conf-1' configuration
   Trying 'kernel-1' kernel subimage
[...]
   Loading Device Tree to 46f49000, end 46f5bfff ... OK
Starting kernel ...
```

DTO Hands-on 1/2

fitImage source format

```
$ cat fit-image.its
/dts-v1/;
/ {
    description = "Linux kernel and FDT blob";
    images {
        kernel-1 {
            description = "Linux kernel";
            data = /incbin/("./tmp/deploy/images/qemuarm/zImage");
            type = "kernel";
            arch = "arm";
            os = "linux";
            compression = "none";
            load = <0x40008000>;
            entry = <0x40008000>;
            hash-1 {
                algo = "crc32";
            };
        };
        fdt-1 {
            description = "Flattened Device Tree blob";
            data = /incbin/("./qemu.dtb");
            type = "flat_dt";
            arch = "arm";
            compression = "none";
            hash-1 {
                algo = "md5";
            };
        };
    };
};
```

DTO Hands-on 1/2

fitImage source format

```
/ {  
[...]  
  
    description = "Linux kernel and FDT blob";  
    images {  
        Kernel-1 {  
            ...  
        };  
        Fdt-1 {  
            ...  
        };  
    };  
  
    configurations {  
        default = "conf-1";  
        conf-1 {  
            description = "Boot Linux kernel with FDT blob";  
            kernel = "kernel-1";  
            fdt = "fdt-1";  
            hash-1 {  
                algo = "sha1";  
            };  
        };  
    };  
};
```

DTO Hands-on 1/2

Notes on fitImage

- It is possible to bundle multiple
 - Kernel images, FDTs, firmwares, bitstreams
 - Have multiple configurations
- The `$fdt_high` variable
 - Sets the upper bound memory address for FDT relocation
 - Special value `0xffffffff = -1` means do not relocate FDT
 - Some platforms need FDT close to the kernel binary
- Information about `ulimage` and `fitImage`, “`iminfo`”
- Extracting data from `fitImage` – “`imxtract`”

DTO Hands-on 1/2

Notes on fitImage iminfo

```
=> iminfo 0x100000

## Checking Image at 00100000 ...
FIT image found
FIT description: Linux kernel and FDT blob
Image 0 (kernel-1)
  Description: Linux kernel
  Type: Kernel Image
  Compression: uncompressed
  Data Start: 0x001000ec
  Data Size: 5393824 Bytes = 5.1 MiB
  Architecture: ARM
  OS: Linux
  Load Address: 0x40008000
  Entry Point: 0x40008000
  Hash algo: crc32
  Hash value: bf5547fe
Image 1 (fdt-1)
[...]
```

Default Configuration: 'conf-1'

Configuration 0 (conf-1)

- Description: Boot Linux kernel with FDT blob
- Kernel: kernel-1
- FDT: fdt-1

```
## Checking hash(es) for FIT Image at 00100000 ...
Hash(es) for Image 0 (kernel-1): crc32+
```

DTO Hands-on 1/2

Notes on fitImage configurations

```
=> iminfo 0x100000
## Checking Image at 00100000 ...
FIT image found
FIT description: Linux kernel and FDT blob
Image 0 (kernel-1)
  Description:  Linux kernel
  Type:        Kernel Image
[...]
```

Image 1 (fdt-1)

```
[...]
Default Configuration: 'conf-1'
Configuration 0 (conf-1)
```



```
=> help bootm
bootm - boot application image from memory
```


Usage:

```
bootm [addr [arg ...]]
    - boot application image stored in memory
...
```

For the new multi component uImage format (FIT) addresses must be extended to include component or configuration unit name:

- addr:<subimg_>** - direct component image specification
- addr#<conf_>** - configuration specification

Use iminfo command to get the list of existing component images and configurations.

DTO Hands-on 1/2

OE kernel-fitimage bbclass

- OE can generate fitImage using the kernel-bbclass
- Add the following entries to your machine config:

```
KERNEL_IMAGETYPE = "fitImage"  
KERNEL_CLASSES += "kernel-fitimage"  
KERNEL_DEVICETREE = "your-machine.dtb"
```

- Note that it is possible to use multiple DTs
 - The fitImage bbclass will generate one fitImage configuration per DT entry



Activity Five

Devtool, next steps
Trevor Woerner

devtool

- a collection of tools for working on *recipes*
 - devtool add
 - devtool edit-recipe
 - devtool upgrade
 - devtool update-recipe
 - *etc...*

devtool

- ...*and more!*
 - devtool modify
 - devtool deploy-target
 - devtool undeploy-target
 - devtool build
 - devtool build-image
 - *etc...*

devtool – past presentations

- ELC 2017
 - Using Devtool To Streamline Your Yocto Project Workflow - Tim Orling
 - <https://www.youtube.com/watch?v=CiD7rB35CRE>
- ELC 2017
 - Yocto Project Extensible SDK: Simplifying the Workflow for Application Developers - Henry Bruce
 - <https://www.youtube.com/watch?v=d3xanDJuXRA&t=57S>

devtool – past presentations

- ELC 2018
 - Working with the Linux Kernel in the Yocto Project - Sean Hudson
 - <https://www.youtube.com/watch?v=tZACGS5nQxw>

devtool – past presentations

- YPDD 2018 - ELC
 - Session 3, Devtool 1 - Tim Orling
 - <https://www.youtube.com/watch?v=C-usM6gFVSY>
- YPDD 2018 - ELC
 - Session 7, Devtool 2 - Tim Orling & Henry Bruce
 - https://www.youtube.com/watch?v=UYsqlP_Qt_Q

devtool – documentation

- Yocto Project Reference Manual
 - chapter 8 - *devtool* Quick Reference
 - <https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#ref-devtool-reference>
- Yocto Project Application Development and the Extensible Software Development Kit (eSDK)
 - chapter 2 - Using the Extensible SDK
 - <https://www.yoctoproject.org/docs/current/sdk-manual/sdk-manual.html#sdk-extensible>

devtool – documentation

- Yocto Project Linux Kernel Development Manual
 - section 2.4 - Using *devtool* to Patch the Kernel
 - <https://www.yoctoproject.org/docs/current/kernel-dev/kernel-dev.html#using-devtool-to-patch-the-kernel>

devtool – documentation

```
$ devtool --help
```

```
usage: devtool [--basepath BASEPATH] [--bbpath BBPATH] [-d] [-q]
        [--color COLOR] [-h]
        <subcommand> ...
```

OpenEmbedded development tool

options:

<code>--basepath BASEPATH</code>	Base directory of SDK / build directory
<code>--bbpath BBPATH</code>	Explicitly specify the BBPATH, rather than getting it from the metadata
<code>-d, --debug</code>	Enable debug output
<code>-q, --quiet</code>	Print only errors
<code>--color COLOR</code>	Colorize output (where COLOR is auto, always, never)
<code>-h, --help</code>	show this help message and exit

subcommands:

Beginning work on a recipe:

<code>add</code>	Add a new recipe
------------------	------------------

...

devtool – documentation

```
$ devtool add --help
```

```
usage: devtool add [-h] [--same-dir | --no-same-dir] [--fetch URI]
                  [--fetch-dev] [--version VERSION] [--no-git]
                  [--srcrev SRCREV | --autorev] [--srcbranch SRCBRANCH]
                  [--binary] [--also-native] [--src-subdir SUBDIR]
                  [--mirrors] [--provides PROVIDES]
                  [recipeName] [srcTree] [fetchUri]
```

Adds a new recipe to the workspace to build a specified source tree. Can optionally fetch a remote URI and unpack it to create the source tree.

arguments:

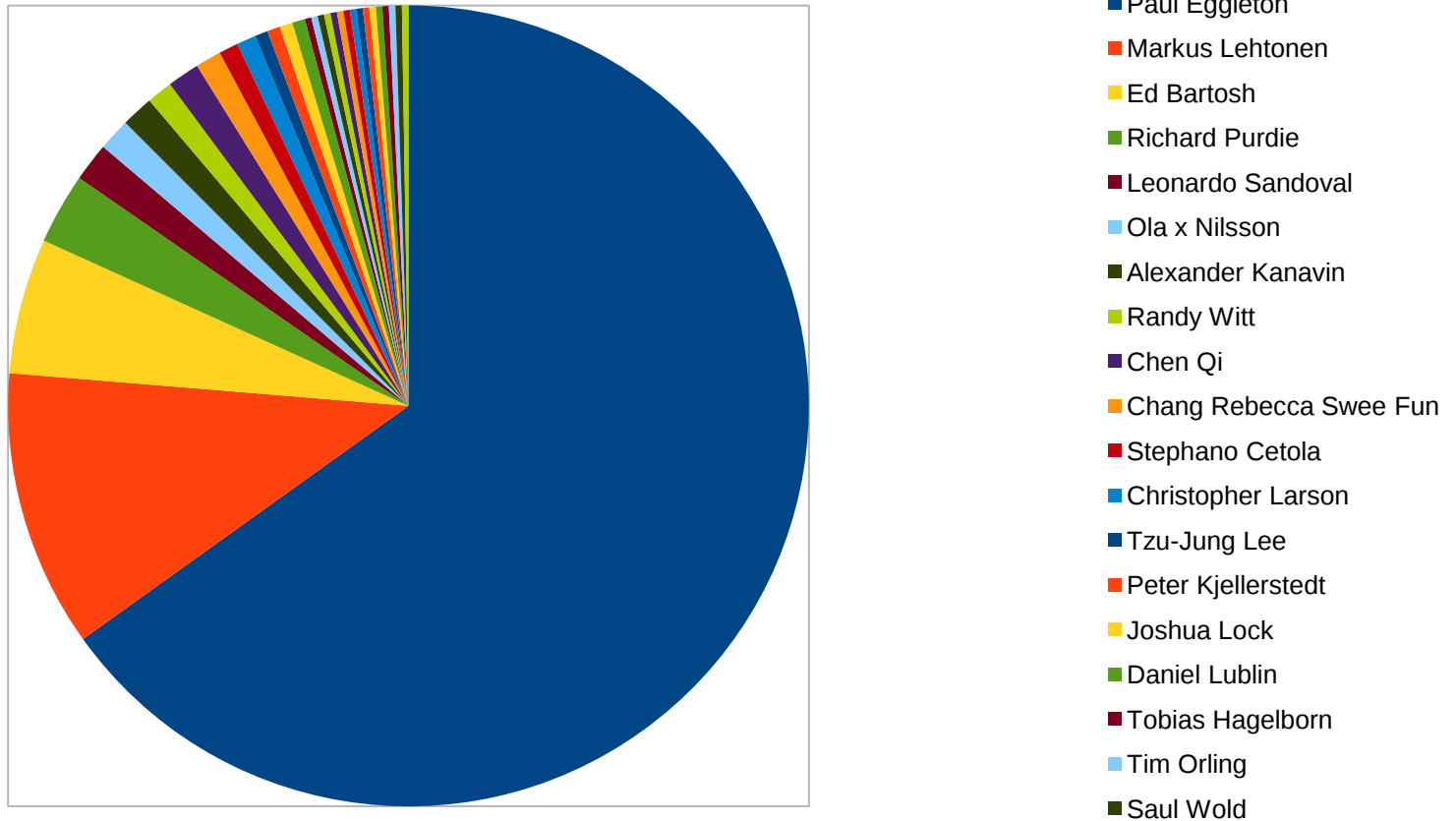
recipeName	Name for new recipe to add (just name - no version, path or extension). If not specified, will attempt to auto-detect it.
srcTree	Path to external source tree. If not specified, a subdirectory of /z/ypdd/2018-10-devtool/my-class/poky/build/workspace/sources will be used.
fetchUri	Fetch the specified URI and extract it to create the source tree

options:

-h, --help	show this help message and exit
------------	---------------------------------

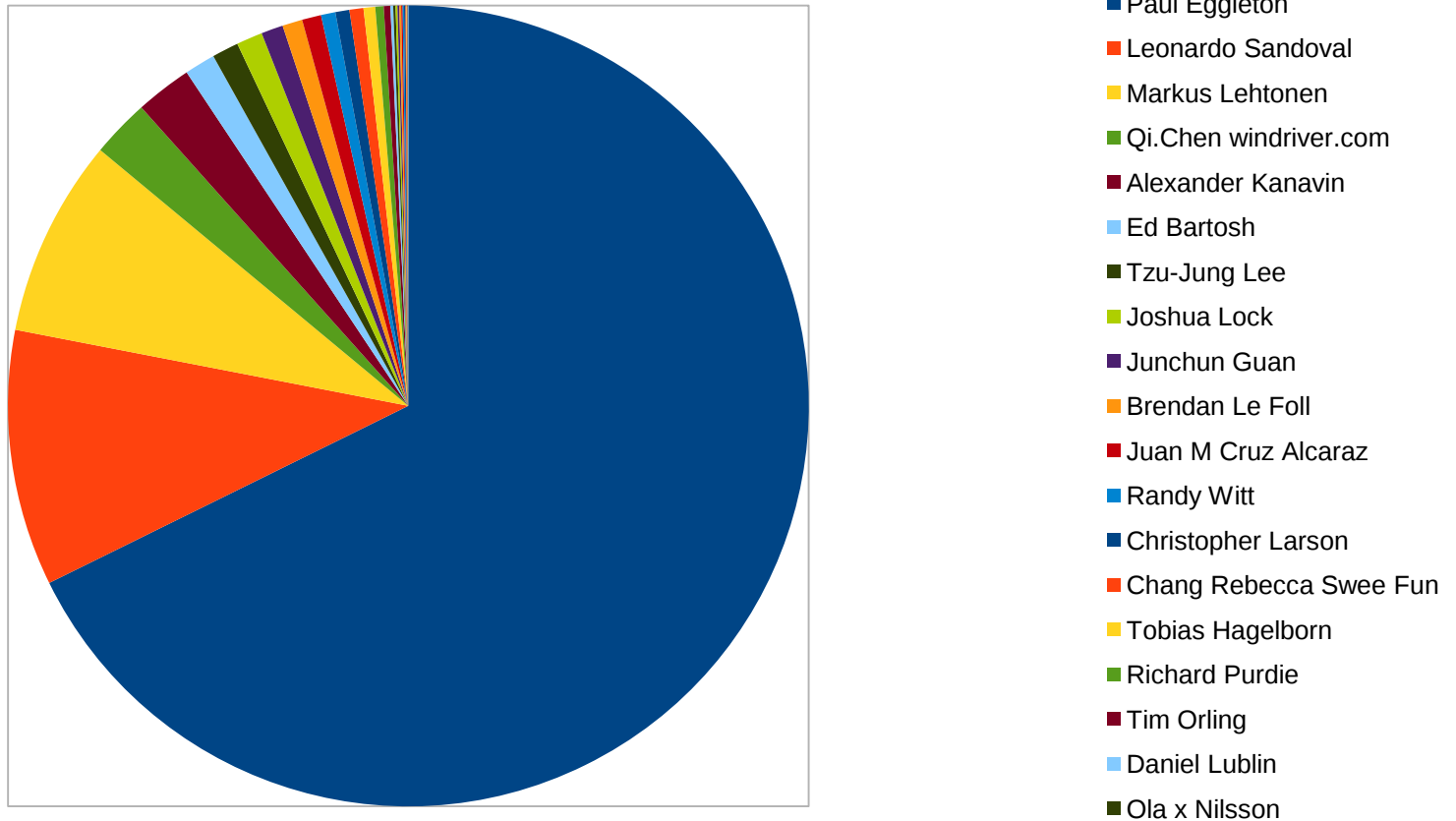
...

devtool development – contributors – by commits



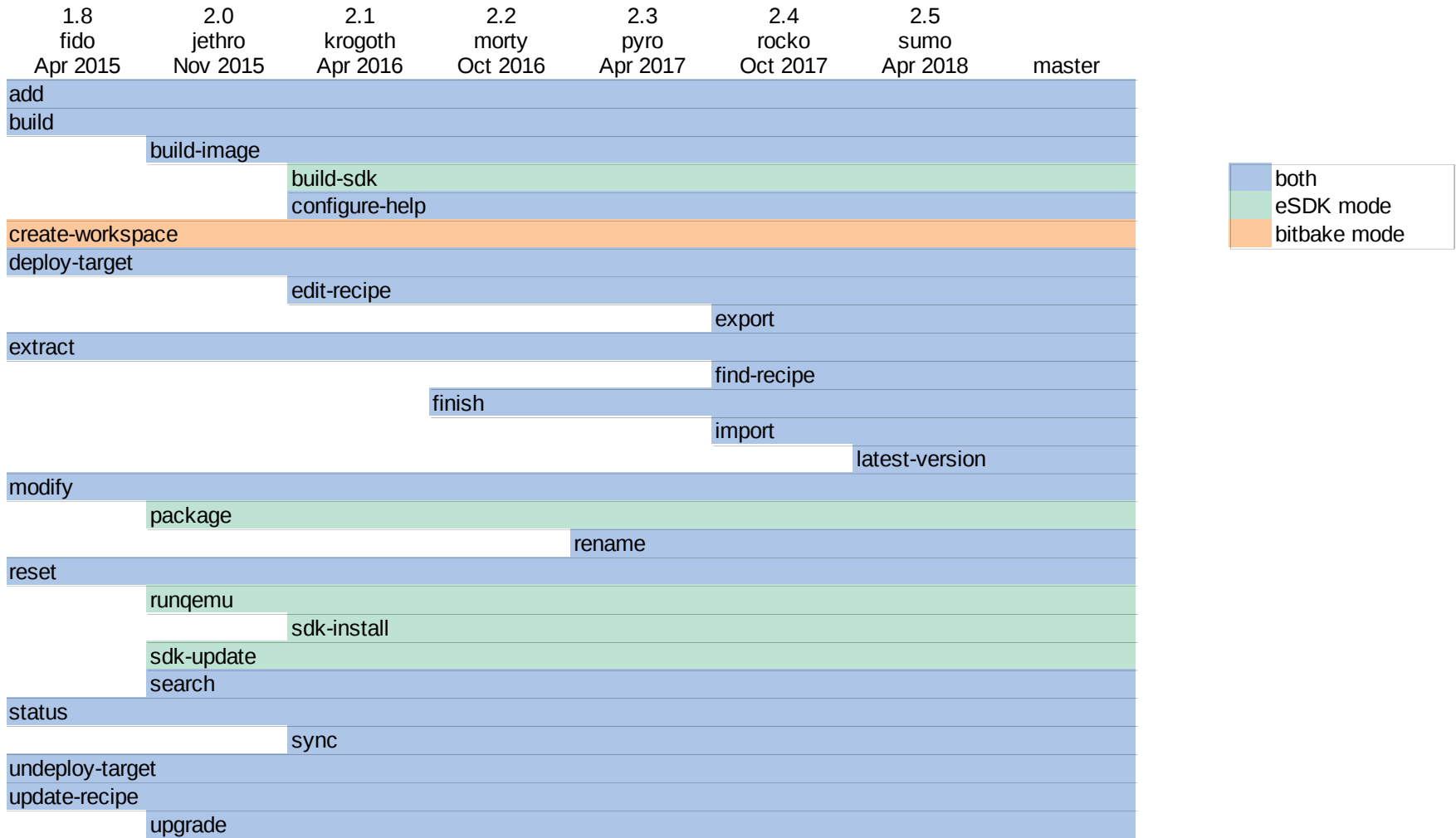
** as of Oct 15,
2018*

devtool development – contributors – by lines

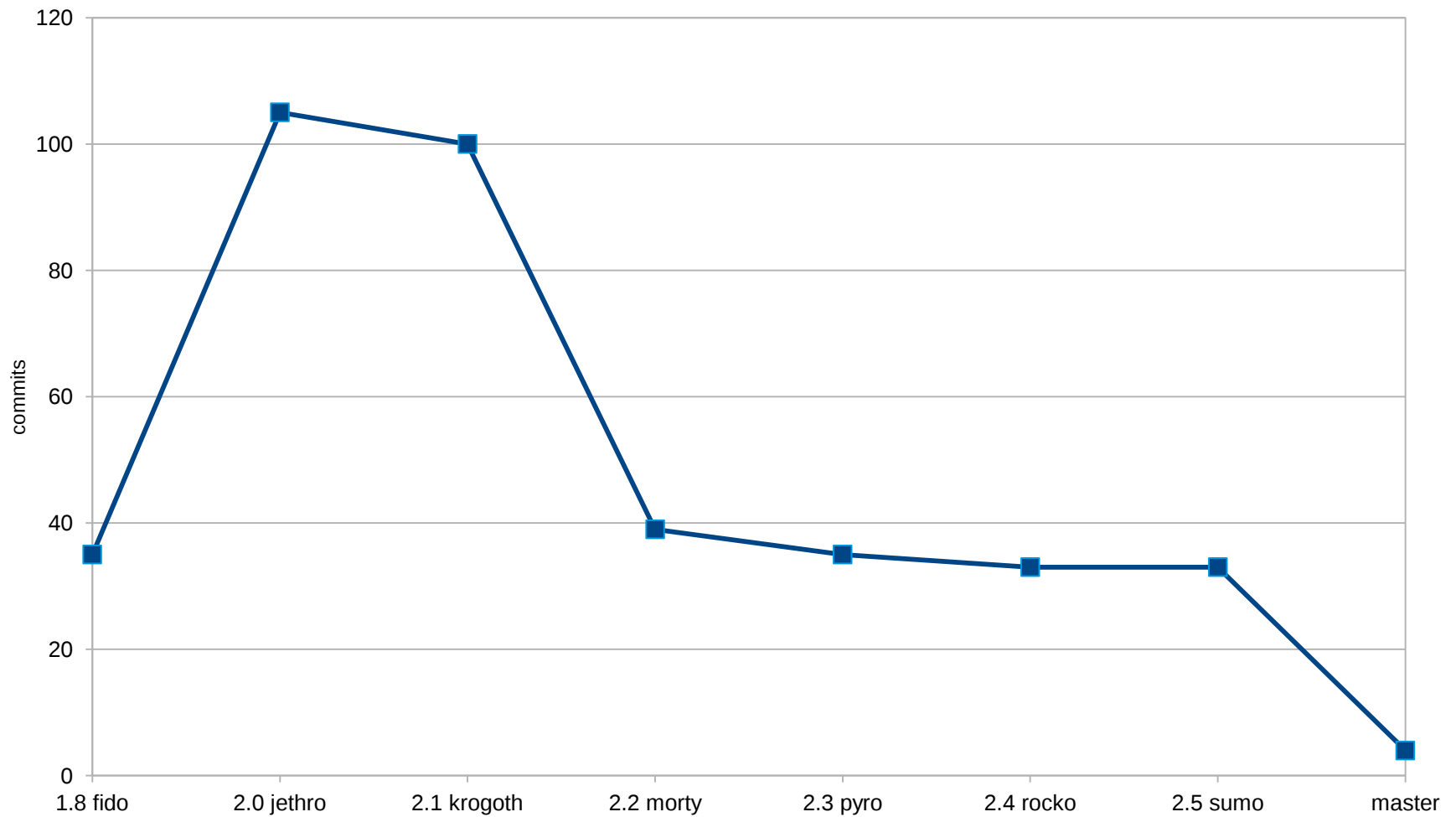


** as of Oct 15,
2018*

devtool development - functionality



devtool development – commits



devtool – modes

- devtool runs in two modes
 - when run inside an eSDK: “eSDK mode”
 - when run outside an eSDK: “bitbake mode”

devtool – mode commands

- bitbake mode

- add
- build
- build-image
- configure-help
- **create-workspace**
- deploy-target
- edit-recipe
- export
- extract
- find-recipe
- finish
- import
- latest-version
- modify
- rename
- reset
- search
- status
- sync
- undeploy-target
- update-recipe
- upgrade

- eSDK mode

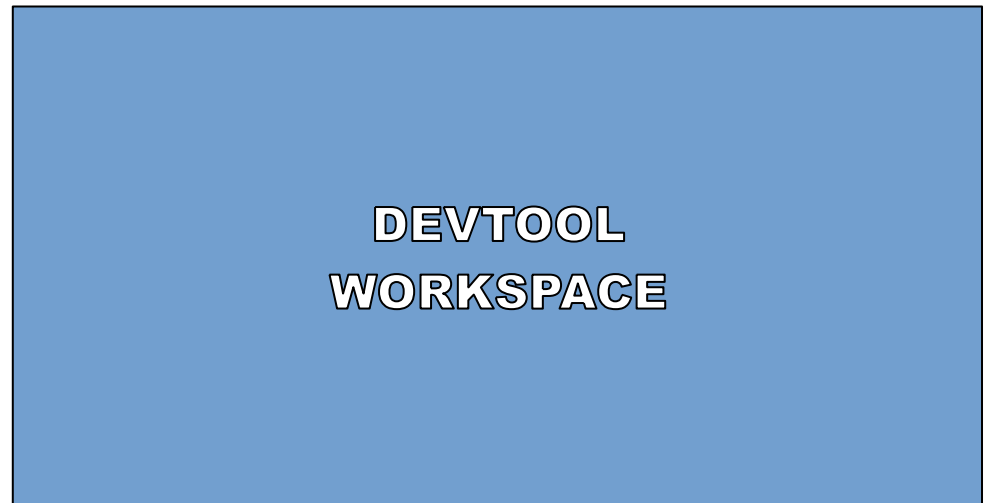
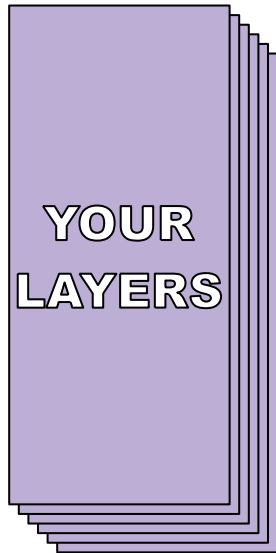
- add
- build
- build-image
- **build-sdk**
- configure-help
- deploy-target
- edit-recipe
- export
- extract
- find-recipe
- finish
- import
- latest-version
- modify
- **package**
- rename
- reset
- **runqemu**
- **sdk-install**
- **sdk-update**
- search
- status
- sync
- undeploy-target
- update-recipe
- upgrade

devtool – mode commands

- why does eSDK mode get all the extra features?
 - because an eSDK doesn't have *bitbake* or *scripts/*
 - *devtool* is the cornerstone of the eSDK

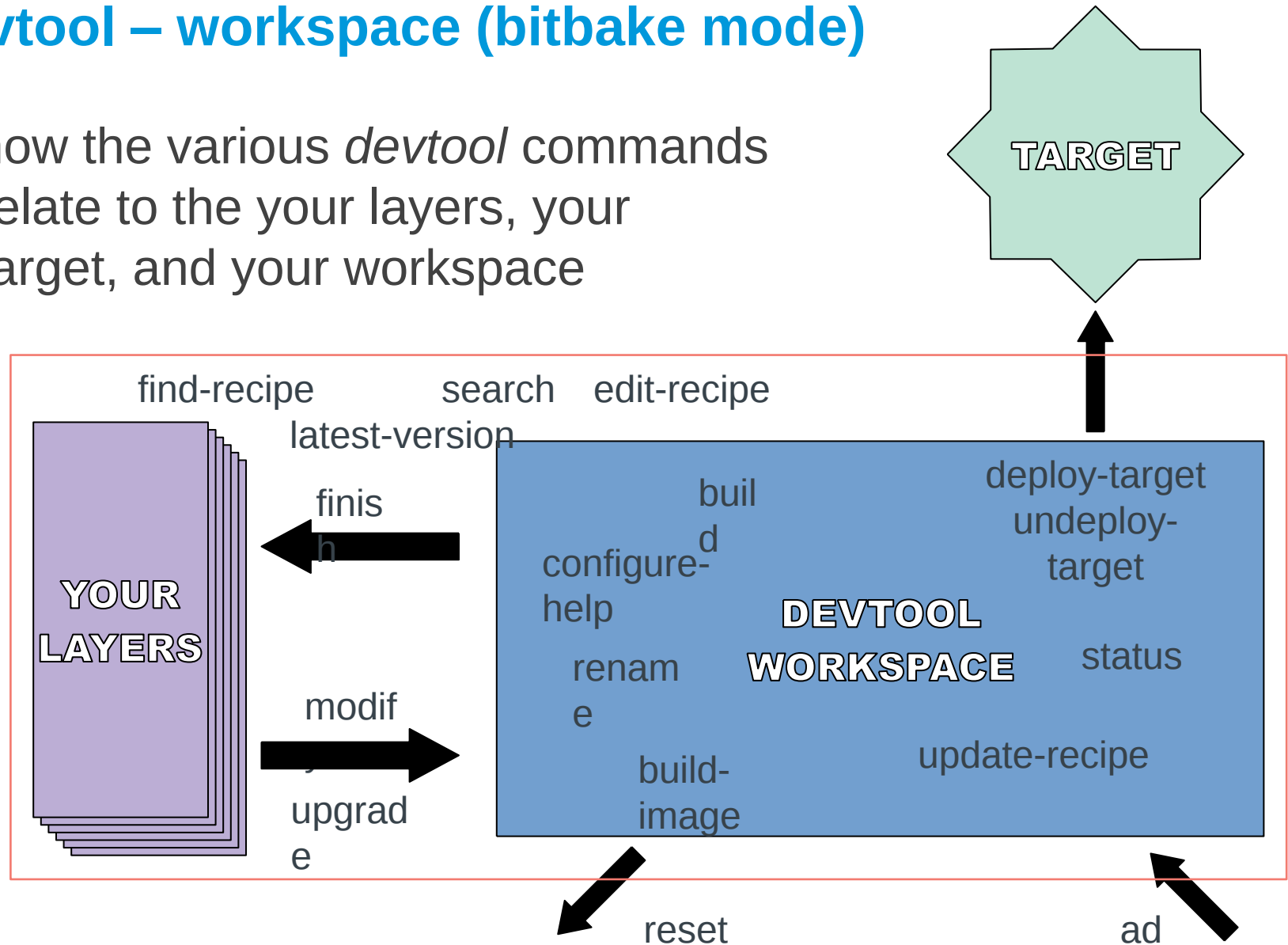
devtool – workspace

- a separate environment (layer) in which to work on recipes, sources, patches

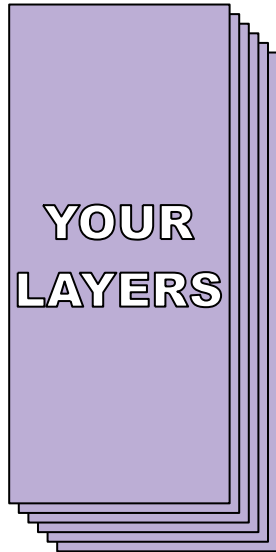


devtool – workspace (bitbake mode)

- how the various *devtool* commands relate to the your layers, your target, and your workspace

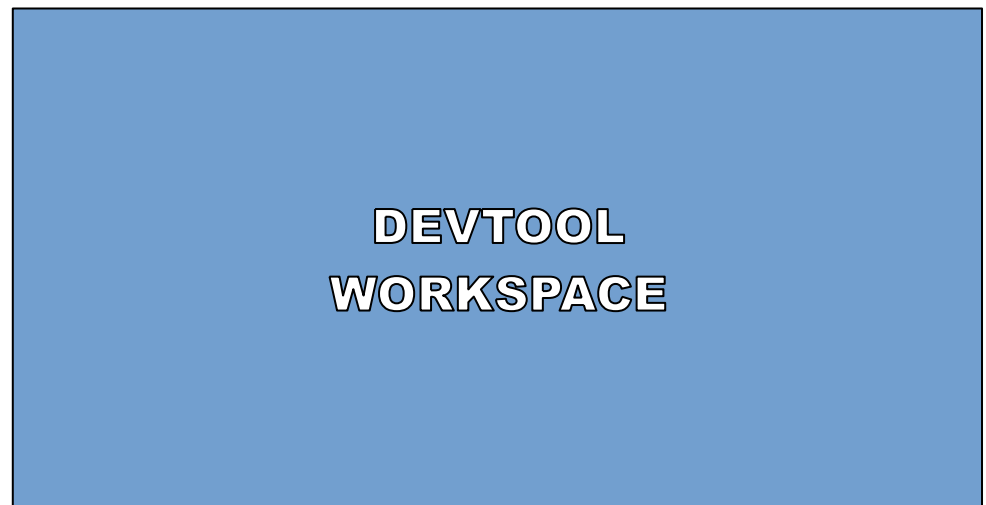
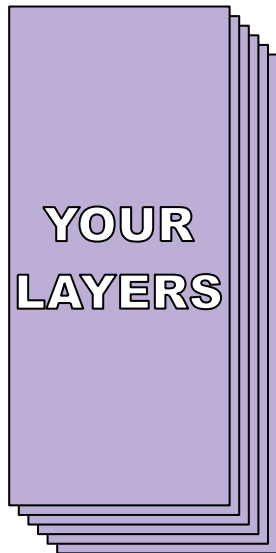


devtool – multiple targets?

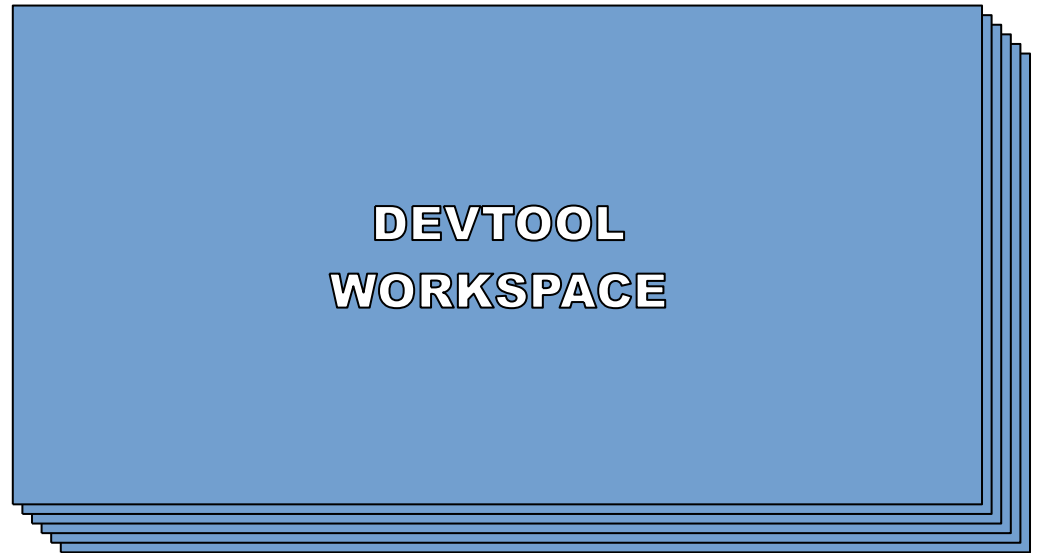
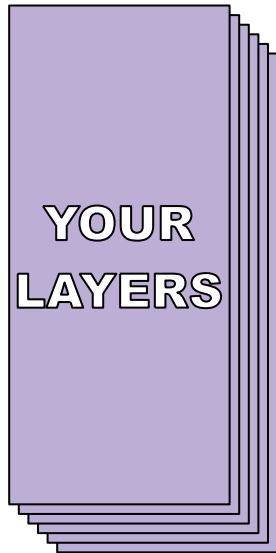


devtool – multiple targets?

- yes
- specify target's IP with un/deploy-target

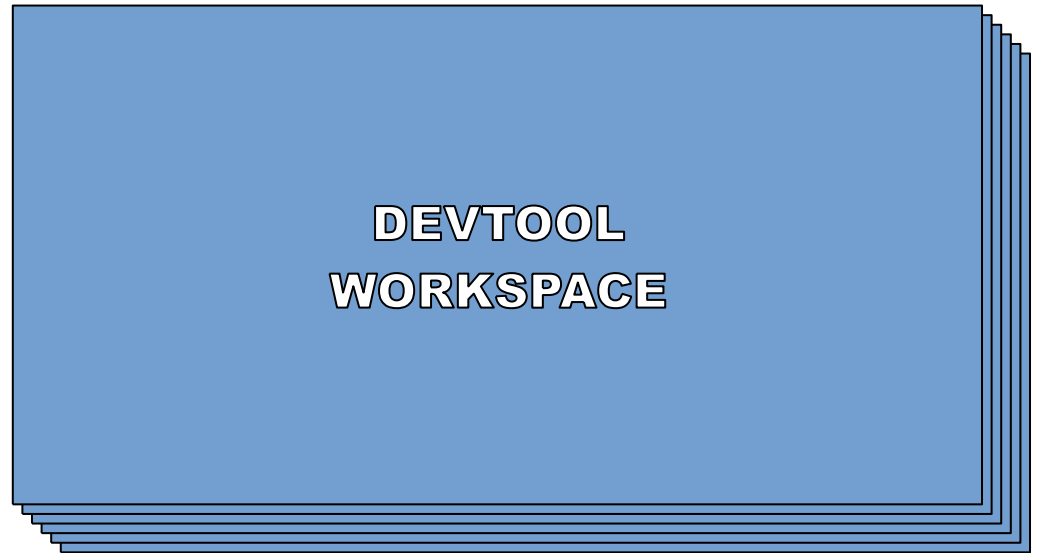
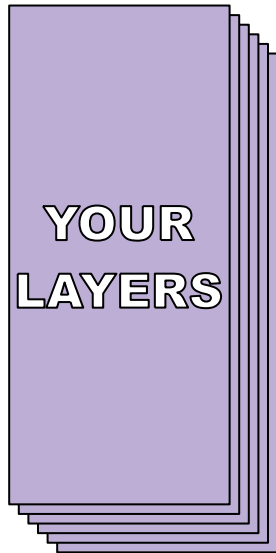


devtool – multiple workspaces?



devtool – multiple workspaces?

- technically: yes (i.e. no errors)
- practically: no (the next replaces the previous, so there's only ever one)



devtool – setup

```
$ edit/create ~/.ssh/config
```

```
Host qemu
```

```
    User root
```

```
    Hostname localhost
```

```
    Port 2222
```

```
    StrictHostKeyChecking no
```

```
    UserKnownHostsFile /dev/null
```

```
$ mkdir devtool; cd devtool
```

```
$ git clone -b sumo git://git.yoctoproject.org/poky
```

```
$ . poky/oe-init-build-env build
```

```
$ edit conf/local.conf
```

```
    MACHINE = "qemuarm"
```

```
    DL_DIR = "/scratch/downloads"
```

```
    SSTATE_DIR = "/scratch/sstate-cache"
```

```
    INHERT += "buildhistory"
```

```
    IMAGE_INSTALL_append = " openssh"
```

```
    WARN_QA_append = " version-going-backwards"
```

```
    ERROR_QA_remove = "version-going-backwards"
```

devtool – setup

```
$ bitbake core-image-base  
$ bitbake-layers create-layer ../meta-foo  
$ bitbake-layers add-layer ../meta-foo  
$ git config --global user.name "name"  
$ git config --global user.email "name@server.com"
```

- open a second ssh connection to the build machine

```
2$ cd devtool  
2$ . poky/oe-init-build-env build  
2$ runqemu slirp nographic serial
```

- do the exercises in the first connection, work on the target in the second connection

devtool – getting started

```
$ devtool add \  
    https://nano-editor.org/dist/v3/nano-3.0.tar.xz
```

- implicitly creates workspace (if it doesn't already exist)
- guesses the recipe name “nano” (correctly!)
- looks at the source and determines it's an autotooled project (true! and pkgconfig and gettext)
- guesses at DEPENDS (correctly!)
- creates a “rough” recipe

```
$ devtool status  
$ devtool find-recipe nano  
$ devtool edit-recipe nano
```

devtool – getting started

- let's see if it builds

```
$ devtool build nano
```

- it works!

devtool – what goes in a workspace?

- the things on which you are working:
 - recipes
 - patches
 - sources
 - etc...

```
$ tree -d workspace
```

- ...except sources can be, optionally, outside the workspace

devtool – let's see nano run

- examine `buildhistory/images/qemuarm/glibc/core-image-base/installed-packages.txt`
 - verify there's no “nano” package
- in the terminal running qemu, log in and verify there's no nano

```
root@qemuarm# nano
-sh: nano: command not found
```

- send nano to target

```
$ devtool deploy-target nano qemu
```

- now nano runs

devtool – let's see nano run

- build an entire image

```
$ devtool build-image core-image-base
...
NOTE: Building image core-image-base with the following
additional packages: nano
...
```

- examine `buildhistory/...../installed-packages.txt`
 - now there is a “nano” package
- why not just use “`bitbake core-image-base`”?
 - nano package not automatically added
 - devtool makes assumptions

devtool – upgrade

- try upgrading nano

```
$ devtool upgrade -V 3.1 nano  
ERROR: recipe nano is already in your workspace
```

- we need to move recipe to layers before upgrade
 - preferably our own (meta-foo)

```
$ devtool finish nano ../meta-foo  
ERROR: Source tree is not clean:  
...
```

- this is not a problem we introduced

```
$ devtool finish -f nano ../meta-foo
```

devtool – upgrade

```
$ devtool upgrade -V 3.1 nano  
$ devtool build nano  
$ devtool deploy-target nano qemu
```

devtool deploy-target - dive in

- is it okay to re-deploy a second time without cleaning up the first deploy?
 - yes... usually
- on the target

```
root@qemuarm# cd /
root@qemuarm# ls -a
...
.devtool
...
root@qemuarm# cd .devtool
root@qemuarm# ls -l
-rw-r--r--      1 root      root           4969 Oct 20 06:03
nano.list
```

devtool deploy-target - dive in

- nano.list is created by devtool, per package, when it deploys to the target
- examine poky/scripts/lib/devtool/deploy.py for all the answers
 - it creates a script that is copied to target
 - preserves any files that would be clobbered
 - generates a list of files being deployed, so they can be undeployed
 - deploying starts by undeploying (same recipe name)

devtool deploy-target - dive in

- undeploy, and verify nano is removed from target, and the plumbing is also removed

```
$ devtool undeploy-target nano qemu
```

```
root@qemuarm# ls -a /
```

- remember to finish and cleanup

```
$ devtool finish -f nano ../meta-foo  
$ rm -fr workspace/sources/nano
```

devtool - floating devtool commands

- some devtool commands don't care whether the recipe is in the workspace or the layers

```
$ devtool status
```

```
NOTE: No recipes currently in your workspace
```

```
$ devtool edit-recipe bash
```

```
(works)
```

```
$ devtool latest-version bash
```

```
NOTE: Current version: 4.4.12
```

```
NOTE: Latest version: 4.4.18
```

```
$ devtool find-recipe bash
```

```
$ devtool search bash
```

devtool - multiple workspaces

- let's look at some devtool plumbing

```
$ cat conf/devtool.conf
[General]
workspace_path = /home/ilab01/devtool/build/workspace

[SDK]
target_basename = core-image-base

$ tail -3 conf/bblayers.conf
/home/ilab01/devtool/meta-foo \
/home/ilab01/devtool/build/workspace \
"
```

devtool - multiple workspaces

- create a new workspace

```
$ devtool create-workspace ws2
$ head -2 conf/devtool.conf
[General]
workspace_path = /home/ilab01/devtool/build/ws2

$ tail -3 conf/bblayers.conf
/home/ilab01/devtool/meta-foo \
/home/ilab01/devtool/build/ws2 \
"
```

- the first one disappears

devtool - creating a patch

- use-case? patches can be needed to
 - add/remove functionality
 - reduce size
 - remove dependency/dependencies
 - allow code to be (cross-)compiled

devtool - creating a patch

```
$ devtool add https://github.com/twoerner/ypdd-elce2018-  
example/archive/v1.0.0.tar.gz  
$ devtool build ypdd-elce2018-example  
$ devtool deploy-target ypdd-elce2018-example qemu
```

```
root@qemuarm# ypdd-elce2018-example  
Hello, world!  
Hello from the library
```

devtool - creating a patch

- edit the code

```
$ pushd workspace/sources/ypdd-elce2018-example  
$ $EDITOR src/ypdd-elce2018-example.c
```

- change from

```
printf("Hello, world!\n");
```

- to

```
printf("Hello, Edinburgh!\n");
```

devtool - creating a patch

- build, deploy, verify

```
$ popd  
$ devtool build ypdd-elce2018-example  
$ devtool deploy-target ypdd-elce2018-example qemu
```

```
root@qemuarm# ypdd-elce2018-example  
Hello, Edinburgh!  
Hello from the library
```

devtool - creating a patch

- cleanup

```
$ devtool finish ypdd-elce2018-example ../meta-foo
ERROR: Source tree is not clean:
M src/ypdd-elce2018-example.c
```

- oops! but it's nice it didn't clobber or lose my work

```
$ pushd workspace/sources/ypdd-elc2018-example
$ git commit -avs
...
$ popd
$ devtool finish ypdd-elce2018-example ../meta-foo
...
NOTE: Adding new patch 0001-update-salutation.patch
...
$ rm -fr workspace/sources/ypdd-elce2018-example
```

devtool - creating conflict

- now we'll update to a newer release, but the newer release will conflict with our patch

```
$ devtool upgrade ypdd-elce2018-example
...
--2018-10-20 12:16:11-- https://github.com/twoerner/ypdd-
elce2018-example/archive/
Resolving github.com (github.com)... 192.30.253.113,
192.30.253.112
Connecting to github.com (github.com)|192.30.253.113|:443...
connected.
HTTP request sent, awaiting response... 404 Not Found
2018-10-20 12:16:11 ERROR 404: Not Found.

NOTE: Current version: 1.0.0
NOTE: Latest version:
```

- devtool can't figure it out, we need to help it

devtool - creating conflict

```
$ devtool upgrade -V 1.0.1 ypdd-elce2018-example
...
WARNING: Command 'git rebase 99271b82b92d8f5f9ecb31099b78895ba7da84ef' failed:
First, rewinding head to replay your work on top of it...
Applying: update salutation
Using index info to reconstruct a base tree...
M       src/ypdd-elce2018-example.c
Falling back to patching base and 3-way merge...
Auto-merging src/ypdd-elce2018-example.c
CONFLICT (content): Merge conflict in src/ypdd-elce2018-example.c
error: Failed to merge in the changes.
Patch failed at 0001 update salutation
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

You will need to resolve conflicts in order to complete the upgrade.
```

devtool - relieving conflict

- keep the new, or keep the old?
 - keep the new

```
$ pushd workspace/sources/ypdd-elce2018-example  
$ $EDITOR src/ypdd-elce2018-example.c
```

devtool - relieving conflict

- from

```
12 <<<<<<< 99271b82b92d8f5f9ecb31099b78895ba7da84ef
13          /* a meaningful comment */
14          printf("Hello, world!\n");
15 =====
16          printf("Hello, Edinburgh!\n");
17 >>>>>>> update salutation
```

- to

```
12          /* a meaningful comment */
13          printf("Hello, Edinburgh!\n");
```

devtool - relieving conflict

```
$ git add src/ypdd-elce2018-example  
$ git rebase --continue  
Applying: update salutation  
$ popd  
$ devtool update-recipe ypdd-elce2018-example
```

- inspect recipe updates

devtool - relieving conflict

```
$ devtool finish ypdd-elce2018-example ../meta-foo
$ tree ../meta-foo
../meta-foo/
...
├── recipes-nano
│   └── nano
│       └── nano_3.1.bb
└── recipes-ypdd-elce2018-example
    ├── ypdd-elce2018-example
    │   ├── ypdd-elce2018-example
    │   └── 0001-update-salutation.patch
    └── ypdd-elce2018-example_1.0.1.bb
```

- considering there's *devtool finish*, how useful is *devtool update-recipe*?

devtool - modify

- take existing recipe from layers
- move recipe to workspace
- unpack sources into workspace
- edit recipe or sources

devtool - eSDK Mode

- the eSDK includes many improvements over the SDK
- combine everything of a regular SDK with all the functionality we've been looking at that is provided by devtool

Questions



Activity Six

Licensing 2.0

Beth Flanagan, Paul Barker

Who Are We?



Togán Labs Ltd

- Ireland based Embedded Linux Consultancy
- Developers of Oryx Linux
- OpenChain Partner, strong focus on license compliance

Beta Five Ltd

- Nottingham, UK based
- Open Source Consultancy
- Linux-based projects from Embedded to Cloud



Contact details

- For any follow up questions or enquiries
- Beth Flanagan, Togán Labs Ltd
 - pidge@toganlabs.com
- Paul Barker, Beta Five Ltd
 - paul@betafive.co.uk

Overview

- **Mirror creation**
- **Providing license manifests & text**
- **Providing recipes**

How downloads work in bitbake

- Sources are fetched into the downloads directory
- Git sources are cloned into bare local repositories
 - Other version control systems are handled similarly
- For each successful fetch a `‘.done’` file is created

Why create a mirror?

- **Best approach for open source distros & BSPs**
- **Makes life easy for downstream Yocto users**
- **Saves you from disappearing sources**

Generating mirror tarballs

- Bitbake supports mirroring git sources as a tarball of the bare repository
 - Again, works similarly for other version control systems
- We can create these tarballs automatically during fetch
 - Set the following variable in local.conf or your distro conf file:
 - `BB_GENERATE_MIRROR_TARBALLS = "1"`

Ensuring downloads is populated

- Need to be careful here, sources may not be re-downloaded if a recipe is built from sstate
 - This applies even if downloads is empty
- Must explicitly run the fetch task for all recipes in our image, SDK or other targets
- Thankfully this can be done via the following methods:
 - **2.5 “sumo” or later:** `bitbake <target> --runall=fetch`
 - **2.4 “rocko” or earlier:** `bitbake <target> -c fetchall`

Collecting mirror files

- We don't need the '.done' files in our mirror
- We also don't need the uncompressed bare git repositories and similar directories for other version control systems
- We use the following magic:

```
mkdir -p mirror
for f in `find downloads -maxdepth 1 -type f -not -
name *.done`; do ln -f $f mirror/`basename $f`; done
```

- These hard links save space but are easy to copy

Serving your mirror

- **Internally**
 - **Local directory**
 - **NFS share**
- **Publically**
 - **HTTP server**

Using the mirror

- Local path:

```
PREMIRRORS_prepend = " \  
ftp://.*/.*      file://${TOPDIR}/mirror/ \  
http://.*/.*     file://${TOPDIR}/mirror/ \  
https://.*/.*    file://${TOPDIR}/mirror/ \  
git://.*/.*      file://${TOPDIR}/mirror/ "
```

- Public mirror:

```
PREMIRRORS_prepend = " \  
ftp://.*/.*      https://example.com/mirror/ \  
http://.*/.*     https://example.com/mirror/ \  
https://.*/.*    https://example.com/mirror/ \  
git://.*/.*      https://example.com/mirror/ "
```

Testing your mirror

- **Set the following in local.conf:**
 - `BB_FETCH_PREMIRRORONLY = "1"`
- **The build will then use only the configured mirror**

The own-mirrors class

- Intended for local testing only
- You can set the following in local.conf:
 - `INHERIT += "own-mirrors"`
`SOURCE_MIRROR_URL =`
`"https://example.com/mirror/"`
- Do not use this in a distro conf as it supports only one `SOURCE_MIRROR_URL` value

License manifests

- Useful to have a simple list of packages installed and their licenses
- This is created automatically during an image build
- See `tmp/deploy/licenses/<image>-<machine>-<timestamp>`
- For example:
- `tmp/deploy/licenses/core-image-base-qemux86-20180926120707/`

License manifests (2)

- **Files created:**
 - `package.manifest`
 - **Simple list of installed packages**
 - `license.manifest`
 - **Packages, versions, recipe names and licenses**
 - `image_license.manifest`
 - **As above for dependencies not directly installed in the image (e.g. bootloader)**

License Text

- For each recipe you will also find a directory in `tmp/deploy/licenses`.
- This contains license texts
- Also contains a `recipeinfo` file summarising the license and recipe version

Including license text in images

- Simple way to ensure end users receive license text
- In `local.conf` or a distro conf you can set:
 - `COPY_LIC_DIRS = "1"`
 - Places license text for each package into `/usr/share/common-licenses`
 - `COPY_LIC_MANIFEST = "1"`
 - Places previously discussed `license.manifest` into `/usr/share/common-licenses`

Including license text in images (2)

- **One caveat...**
- `COPY_LIC_DIRS` **and** `COPY_LIC_MANIFEST` **only cover packages installed during image creation**
- **Licenses for packages installed via on-target package management are not handled by these methods**

Creating license packages

- **Another variable you can set:**
 - `LICENSE_CREATE_PACKAGE = "1"`
- **For each recipe this creates a `${PN}-lic` package**
 - **E.g.** `busybox-lic`
- **Adds this as an RRECOMMENDS for the base package**
- **Installs licenses into `/usr/share/licenses/${PN}`**
 - **E.g.** `/usr/share/licenses/busybox`

Providing recipes

- The archiver can be used to provide recipes
 - Creates tarball of the bb file, bbappends & includes
- However, this makes it difficult for users to rebuild images
- It can be argued from the GPL that providing full layers is required
 - “scripts used to control compilation and installation”
 - I’m not a lawyer!

Providing recipes (2)

- **The best way to handle this is to release your layers**
- **Also ensure you snapshot bitbake and third party layers used to build release images**
- **Recommend you also provide bblayers.conf, local.conf and any other customisation**

Releasing your layer

- **Releasing publically as an open source layer is easiest**
 - You can add your layer to <http://layers.openembedded.org/>
- **However, you can also release privately to customers**
 - **Give people a source archive or a download link with your product or images**

Providing the correct versions

- **Please don't just point people at a layer repository or branch**
- **Make sure they get the same exact versions of bitbake and metadata which was used to build your image**
- **Many ways to do this**
 - **Tarball**
 - **Git submodules**
 - **Repo tool**

Avoid AUTOREV for releases

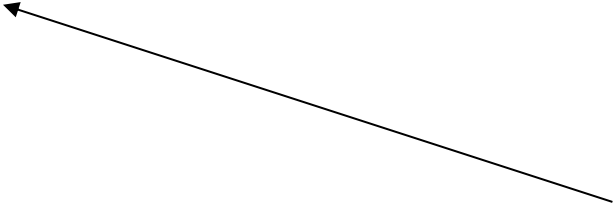
- **Setting** `SRCREV = "${AUTOREV}"` **can be great in development**
- **Terrible for releases**
- **People receiving your layer may need to rebuild months or years later and could get a different git commit**
- **Always explicitly set SRCREV when building releases**

Don't be clever

```
DESCRIPTION = "Node.js modules"
LICENSE = "MIT & ISC & Apache-2 & FIPL-1.0 & BSD-2-  
Clause"
DEPENDS = "nodejs-native glfw glew cairo pango jpeg  
libpng"
DEPENDS_class-native = "nodejs-native"
PROVIDES = "nodejs-modules"
PR = "r2"
S = "${WORKDIR}/${PN}-${PV}"
PD= "${PN}-${PV}/packages"
require packages.inc
```

Don't be clever

```
DESCRIPTION = "Node.js modules"  
LICENSE = "MIT & ISC & Apache-2 & FIPL-1.0 & BSD-2-  
Clause"  
DEPENDS = "nodejs-native glfw glew cairo pango jpeg  
libpng"  
DEPENDS_class-native = "nodejs-native"  
PROVIDES = "nodejs-modules"  
PR = "r2"  
S = "${WORKDIR}/${PN}-${PV}"  
PD= "${PN}-${PV}/packages"  
require packages.inc
```



Wait? Wot?

Don't be clever

```
SRC_URI+= "http://registry.npmjs.org/put/-/put-0.0.6.tgz;name=0017put;unpack=yes;downloadfilename=put-0.0.6.tgz;subdir=${PD}/0017-put-0.0.6"  
LIC_FILES_CHKSUM += "file://real/put-  
0.0.6/package/LICENSE;md5=b2d989bc186e7f6b418a5fdd5cc0b56b"
```

```
SRC_URI+= "http://registry.npmjs.org/sax/-/sax-1.2.1.tgz;name=0018sax;unpack=yes;downloadfilename=sax-1.2.1.tgz;subdir=${PD}/0018-sax-1.2.1"  
LIC_FILES_CHKSUM += "file://real/sax-  
1.2.1/package/LICENSE;md5=326d5674181c4bb210e424772c60fa80"
```

```
SRC_URI+= "http://registry.npmjs.org/through/-/through-2.3.8.tgz;name=0019through;unpack=yes;downloadfilename=through-2.3.8.tgz;subdir=${PD}/0019-  
through-2.3.8"  
LIC_FILES_CHKSUM += "file://real/through-  
2.3.8/package/readme.markdown;md5=6ff48d70322f9b54b7f36536954bca06"  
LIC_FILES_CHKSUM += "file://real/through-  
2.3.8/package/LICENSE.APACHE2;md5=ffcf739dca268cb0f20336d6c1a038f1"  
LIC_FILES_CHKSUM += "file://real/through-  
2.3.8/package/LICENSE.MIT;md5=e0f70a42adf526e6f5e605a94d98a420"
```

```
SRC_URI+=
```

Trust but verify

- meta-license-tools + fossup + fossology
- Patched archiver scans
- SLOW!!! But finds issues
- A lot of knowledge needed about what you're actually distributing.

```
USER_CLASSES += "license archiver"
COPYLEFT_LICENSE_INCLUDE = "GPL* AGPL* LGPL* MPL*"
COPYLEFT_LICENSE_EXCLUDE = "CLOSED Proprietary"
ARCHIVER_MODE[src] = "patched"
ARCHIVER_MODE[diff] = "0"
ARCHIVER_MODE[dumpdata] = "0"
ARCHIVER_MODE[recipe] = "1"
COPYLEFT_RECIPE_TYPES = "target"
INHERIT += "fossology"
VM_SPRINT_NUMBER = "054"
```

Lets do a FOSSOLOGY CLEARANCE!



Activity Seven

Device Trees 2.0

Marek Vasut

Device Tree

The Device Tree

- Data structure describing hardware
- Usually passed to OS to provide information about HW topology where it cannot be detected/probed
- Tree, made of named nodes and properties
 - Nodes can contain other nodes and properties
 - Properties are a name-value pair
 - See https://en.wikipedia.org/wiki/Device_tree
- DT can contain cycles by means of phandles
 - phandles provide simple references to device definitions (e.g. “<&L2>” = level 2 cache definition)
 - phandles can be used to reference objects in different trees (e.g. use that predefined cache type)

Device Tree Example

- arch/arm/boot/dts/arm-realview-eb-a9mp.dts

```
/dts-v1/;
#include "arm-realview-eb-mp.dtsi"
/ {
    model = "ARM RealView EB Cortex A9 MPCore";
    [...]
    cpus {
        #address-cells = <1>;
        #size-cells = <0>;
        enable-method = "arm,realview-smp";
        A9_0: cpu@0 {
            device_type = "cpu";
            compatible = "arm,cortex-a9";
            reg = <0>;
            next-level-cache = <&L2>;
        };
    };
    [...]
    &pmu {
        interrupt-affinity = <&A9_0>, <&A9_1>, <&A9_2>, <&A9_3>;
    };
};
```

C-like inheritance

Instance = reg

pHandles ("&")

Problem – Variable hardware

A bit of DT history

- DT started on big machines
 - Hardware was mostly static
 - DT was baked into ROM, optionally modified by bootloader
- DT was good, so it spread
 - First PPC, embedded PPC, then ARM ...
- There always was slightly variable hardware
 - Solved by patching DT in bootloader
 - Solved by carrying multiple DTs
 - Solved by co-operation of board files and DT
 - ^ all that does not scale

Problem – Variable hardware – 201x edition

DT today

- **Come 201x, variable cardware became easy to make:**
 - Cheap devkits with hats, lures, capes, ...
 - FPGAs and SoC+FPGAs became commonplace ...
 - => Combinatorial explosion of possible HW configurations
- **Solution retaining developers' sanity**
 - Describe only the piece of HW that is being added
 - Combine these descriptions to create a DT for the system
 - Enter DT overlays

Device Tree Overlays

Simple DTO structure

- DT: Data structure describing hardware
- DTO: necessary change(s) to the DT to support particular feature
 - Example: an expansion board, a hardware quirk,...
- Example DTO: vendor='hello', devicetype='dto' (no magic)

```
/dts-v1/;
/plugin/;
/ {
    #address-cells = <1>;
    #size-cells = <0>;
    fragment@0 {
        reg = <0>;
        target-path = "/";
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;
            hello@0 {
                compatible = "hello,dto";
                reg = <0>;
            };
        };
    };
};
```

Overlay at DT root

Must match
kernel module's
compatibility

Advanced DTO example

- Enable USB port, ETH port (over “gmii” channel)

```
/dts-v1/;
/plugin/;
[...]
    fragment@2 {
        reg = <2>;
        target-path = "/soc/usb@fffb40000";
        __overlay__ {
            status = "okay";
        };
    };

    fragment@3 {
        reg = <3>;
        target = <&gmac1>; /* phandle */
        __overlay__ {
            status = "okay";
            phy-mode = "gmii";
        };
    };
[...]
```

Enable this USB port

Enable this Ethernet
port, use “gigabit
media-independent interface”

DTO Hands-on

Practical part

- Use pre-prepared meta-dto-microdemo layer
- meta-dto-demo contains:
 - Kernel patch with DTO loader with ConfigFS interface
 - Kernel config fragment to enable the DTO and loader
 - Demo module
 - Demo DTO source (hello-dto.dts)
 - core-image-dto-microdemo derivative from core-image-minimal with added DTO examples and DTC

DTO Example Layer Tree

```
\-- meta-dto-microdemo
|  |-- conf
|  |  |-- layer.conf
|  |-- recipes-core
|  |  |-- images
|  |  |  |-- core-image-dto-microdemo.bb
|  |-- recipes-kernel
|  |  |-- hello-dto-dto
|  |  |  |-- files
|  |  |  |  |-- hello-dto.dts
|  |  |  |-- hello-dto-dto_0.1.bb
|  |  |-- hello-dto-mod
|  |  |  |-- files
|  |  |  |  |-- COPYING
|  |  |  |  |-- hello-dto.c
|  |  |  |  |-- Makefile
|  |  |-- hello-dto-mod_0.1.bb
|  |-- linux
|  |  |-- files
|  |  |  |-- 0001-ARM-dts-Compile-the-DTS-with-symbols-enabled.patch
|  |  |  |-- 0002-OF-DT-Overlay-configfs-interface-v7.patch
|  |  |-- enable-dtos.cfg
|  |-- linux-yocto_4.12.bbappend
```

" dtc hello-dto-mod hello-dto-dto "

"install -m 0644 *.dts \${D}/lib/firmware/dto/"

Debug messages for module load, remove

" +DTC_FLAGS := -@"

Patch in "overlay-configfs" (*)

**CONFIG_OF_OVERLAY=y
CONFIG_OF_CONFIGFS=y**

DTO Hands-on 1/2

Practical part – Apply DTO from running Linux

- Add meta-dto-demo to bblayers.conf BBLAYERS:

```
$ echo "BBLAYERS += \"/scratch/src/dto/meta-dto-microdemo\" \" \" \
>> conf/bb_layers.conf
$ echo "MACHINE = \"qemuarm\" \" \" >> conf/local.conf
$ echo "SSTATE_DIR = \"/scratch/sstate-cache\" \" \" >> conf/local.conf
$ echo "DL_DIR = \"/scratch/downloads\" \" \" >> conf/local.conf
```

- Rebuild virtual/kernel and core-image-dto-microdemo

```
$ bitbake -c cleansstate virtual/kernel
$ bitbake core-image-dto-microdemo
```

- Adjust runqemu configuration for the ‘virt’ machine (hack)

```
$ sed -i \
    -e "/^qb_machine/      s/.*qb_machine = -machine virt/" \
    -e "/^qb_dtb/          s/.*qb_dtb = /" \
    -e "/^qb_opt_append/   s/.*qb_opt_append = -show-cursor/" \
    tmp/deploy/images/qemuarm/ \
    core-image-dto-microdemo-qemuarm.gemuboot.conf
```

DTO Hands-on 1/2

Practical part – Apply DTO from running Linux

- Start the new image in QEMU (*login: root, no password*)

```
$ runqemu slirp qemuarm nographic
```

- (**CTRL-A x** to quit QEMU)

DTO Hands-on 2/2

- **Compile DTO**

```
$ dtc -I dts -O dtb /lib/firmware/dto/hello-dto.dts > \
/tmp/hello-dto.dtb
```

- **Load DTO**

```
$ mkdir /sys/kernel/config/device-tree/overlays/mydto
$ cat /tmp/hello-dto.dtb > \
/sys/kernel/config/device-tree/overlays/mydto/dtbo
```

- **Confirm DTO was loaded**

```
# ls /proc/device-tree
... hello@0 ...
# ls /sys/kernel/config/device-tree/overlays/mydto
dtbo      path      status
# cat /sys/kernel/config/device-tree/overlays/mydto/status
Applied
#
```

```
# rmdir /sys/kernel/config/device-tree/overlays/mydto
```

DTO Overlay Patch, DTO Workflows

Drawbacks of DTOs in Linux

- **Why is the configs overlay support in a patch?**
 - It is not being accepted into mainline kernel because of the potential security risk (i.e. manufactures accidentally ship it in a production device and do not lock it down)
- **U-Boot can also modify DT and pass the modified DT to Linux**
 - Look at “fdt” command
 - Load DT, point U-Boot to it
 - Use “fdt” command to add/remove/modify nodes and props
 - Boot Linux with the same address to which U-Boot is pointing

DTO Hands-on 2/2

Modify DT in U-Boot manually

- **Create /chosen/bootargs prop with value “hello=kernel”**

```
=> fdt addr $fdtcontroladdr
=> fdt resize
=> fdt print /chosen bootargs
libfdt fdt_getprop(): FDT_ERR_NOTFOUND
=> fdt set /chosen bootargs hello=kernel
=> fdt print /chosen bootargs
bootargs = "hello=kernel"
```

- **Start the kernel with modified QEMU DT**

```
=> bootm 0x100000:kernel-1 - $fdtcontroladdr
## Loading kernel from FIT Image at 00100000 ...
   Trying 'kernel-1' kernel subimage
   ...
Starting kernel ...
[    0.000000] Booting Linux on physical CPU 0x0
[    0.000000] Linux version 4.14.67-yocto-standard (oe-user@oe-host...)
   ...
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages...
[    0.000000] Kernel command line: hello=kernel
[    0.000000] PID hash table entries: 512 (order: -1, 2048 bytes)
```

DTO Hands-on 2/2

Apply DTO in U-Boot manually

- Use DTC to compile DTO

```
dtc -I dts -O dtb \
    meta-dto-microdemo/recipes-kernel/linux/files/fdto.dts \
    -o fdto.dtb
```

- Generate the NOR flash image with U-Boot, zImage and DTO

```
dd if=/dev/zero of=/tmp/test.bin bs=16M count=0 seek=1
dd if=tmp/deploy/images/qemuarm/u-boot.bin of=/tmp/test.bin \
    conv=notrunc
dd if=/tmp/fitImage of=/tmp/test.bin bs=1M seek=1 conv=notrunc
dd if=fdto.dtb of=/tmp/test.bin bs=1M seek=8 conv=notrunc
```

- Start the QEMU

```
$ qemu-system-arm -machine virt -bios /tmp/test.bin -nographic
U-Boot 2018.01 (Oct 14 2018 - 13:36:49 +0000)
DRAM: 128 MiB
Hit any key to stop autoboot: 0
=>
```

DTO Hands-on 2/2

- **Apply DTO to QEMU DT in U-Boot manually**

```
=> help fdt
fdt - flattened device tree utility commands
Usage:
fdt addr [-c] <addr> [<length>]
    - Set the [control] fdt location to <addr>
fdt apply <addr>
    - Apply overlay to the DT
fdt resize [<extrasize>]
    - Resize fdt to size + padding to 4k addr + some
      optional <extrasize> if needed
fdt print <path> [<prop>]
    - Recursive print starting at <path>

=> fdt addr $fdtcontroladdr
=> fdt resize
=> fdt print /chosen
chosen {
    stdout-path = "/p1011@9000000";
};
=> fdt apply 0x800000
=> fdt print /chosen
chosen {
    bootargs = "hello=dto"; // <----- Here is the new node
    stdout-path = "/p1011@9000000";
};
```

DTO Hands-on 2/2

Apply DTO in fitImage to DT in fitImage

- **Generate fitImage with bundled DTO**

```
$ cp meta-dto-microdemo/recipes-kernel/linux/files/\
    fit-image-dto.its .
$ dtc -I dts -O dtb meta-dto-microdemo/recipes-kernel/\
    linux/files/fdto.dts -o fdto.dtb
$ ./tmp/work/x86_64-linux/u-boot-mkimage-native/\
    1_2018.01-r0/git/tools/mkimage \
    -f ./fit-image-dto.its /tmp/fitImage
FIT description: Linux kernel and FDT blob
Created:          Sun Oct 14 14:17:28 2018
Image 0 (kernel-1)
...
Configuration 0 (conf-1)
  Description:    Boot Linux kernel with FDT blob
  Kernel:        kernel-1
  FDT:           fdt-1
                Fdto-1
$ dd if=/dev/zero of=/tmp/test.bin bs=16M count=0 seek=1
$ dd if=tmp/deploy/images/qemuarm/u-boot.bin of=/tmp/test.bin \
    conv=notrunc
$ dd if=/tmp/fitImage of=/tmp/test.bin bs=1M seek=1 conv=notrunc
```

DTO Hands-on 2/2

- FitImage source files extras

```
/dts-v1/;
/ {
...
    fdto-1 {
        description = "Flattened Device Tree overlay blob";
        data = /incbin/("./fdto.dtb");
        type = "flat_dt";
        arch = "arm";
        compression = "none";
        load = <0x44008000>;
        hash-1 {
            algo = "md5";
        };
    };
...
    configurations {
        conf-1 {
            fdt = "fdt-1", "fdto-1";
...
        };
    };
};
```

DTO Hands-on 2/2

- Boot fitImage containing DTO

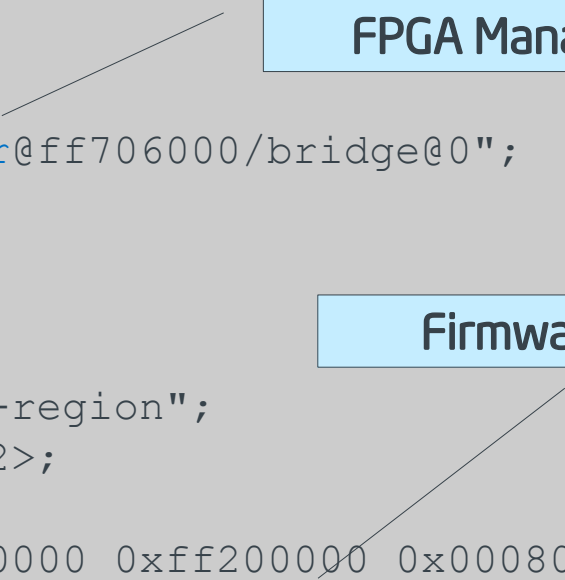
```
$ qemu-system-arm -machine virt -bios /tmp/test.bin -nographic
U-Boot 2018.01 (Oct 14 2018 - 13:36:49 +0000)
=> setenv fdt_high 0x48000000 ; bootm 0x100000
## Loading kernel from FIT Image at 00100000 ...
[...]
Loading fdt from 0x00624f40 to 0x44000000
## Loading fdt from FIT Image at 00100000 ...
Trying 'fdto-1' fdt subimage
Description: Flattened Device Tree overlay blob
Type: Flat Device Tree
Compression: uncompressed
Data Start: 0x00635010
Data Size: 177 Bytes = 177 Bytes
Architecture: ARM
Load Address: 0x44008000
Hash algo: md5
Hash value: 7fd334678b272c17fbfac51bcd9a40c
Verifying Hash Integrity ... md5+ OK
Loading fdt from 0x00635010 to 0x44008000
Booting using the fdt blob at 0x44000000
Loading Kernel Image ... OK
Loading Device Tree to 46f58000, end 46f5cc0a ... OK

Starting kernel ...
[ 0.000000] Booting Linux on physical CPU 0x0
[ 0.000000] Linux version 4.14.67-yocto-standard (oe-user@oe-host) (gcc version 7.3.0 (GCC))...
...
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 32480
[ 0.000000] Kernel command line: hello=dto
[ 0.000000] PID hash table entries: 512 (order: -1, 2048 bytes)
```

DTO encore

- DTOs can be used to operate SoC+FPGA hardware
- Done using FPGA manager in Linux (load firmware into ASIC)

```
fragment@0 {
    reg = <0>;
    /* controlling bridge */
    target-path = "/soc/fpgamgr@ff706000/bridge@0";
    __overlay__ {
        #address-cells = <1>;
        #size-cells = <1>;
        region@0 {
            compatible = "fpga-region";
            #address-cells = <2>;
            #size-cells = <1>;
            ranges = <0 0x00000000 0xff200000 0x00080000>;
            firmware-name = "fpga/bitstream.rbf";
            fpga_version@0 {
                compatible = "vendor,fpgablock-1.0";
                reg = <0 0x0 0x04>;
            };
        };
    };
}
```



FPGA Manager

Firmware file

DTO Overlay Patch, DTO Workflows

- **Example for newbie using DTO's to prepare and debug DT's**
 - Debug the DTO with the manual configs overlay
 - Add the DTO to U-Boot and then debug the hardware and kernel modules
 - Turn the DTO into a pure DT for production

DTO Extra

- **Recommended way to load custom DTO at boot**
 - There are sysvinit, systemd, custom scripts, or add to uboot, however there is no standard for that currently
- **Top debugging techniques, tricks and tips:**
 - The “/proc/device-tree” is the image of the live DT, to check if your overlay was applied properly
 - The configfs interface provides you a status information for each overlay
- **Top common user errors and gothchas**
 - Usually typos in the DT (no verification in DT compiler)
 - Not exact “compatible” match between DTO/kernel module
- **Anything special about DTO's vis-à-vis Yocto Project**
 - Not really, they are orthogonal
 - The “dtc” compiler is part of openembedded-core layer

DTO Extra

- Examples of DTO's in production systems
 - RaspberryPi (hats)
 - Beaglebone (cape manager)
 - Some Intel boards (e.g. via ACPI)
- More on DT at:
 - <https://www.devicetree.org/>
- ePAPR specification of DT:
 - https://elinux.org/images/c/cf/Power_ePAPR_APPROVED_v1.1.pdf
- Contact:
 - Contact: Marek Vasut marek.vasut@gmail.com
 - https://schd.ws/hosted_files/elciotna18/c1/elc-2018.pdf



Activity Eight

Image Size Reduction

Scott Murray

Why bother optimizing distribution size?

- There are embedded products that still benefit from the cost-optimization of reducing RAM and storage footprint
 - e.g. developers interested in using Linux in IoT edge devices
- An increasing need to keep devices up to date means smaller images have a bandwidth and download time advantage, and potentially a reduced security attack surface
- Use cases such as:
 - Small recovery partition images
 - Container images
 - Supporting older hardware (e.g. Tiny Core Linux, <http://distro.ibiblio.org/tinycorelinux/>)

poky-tiny

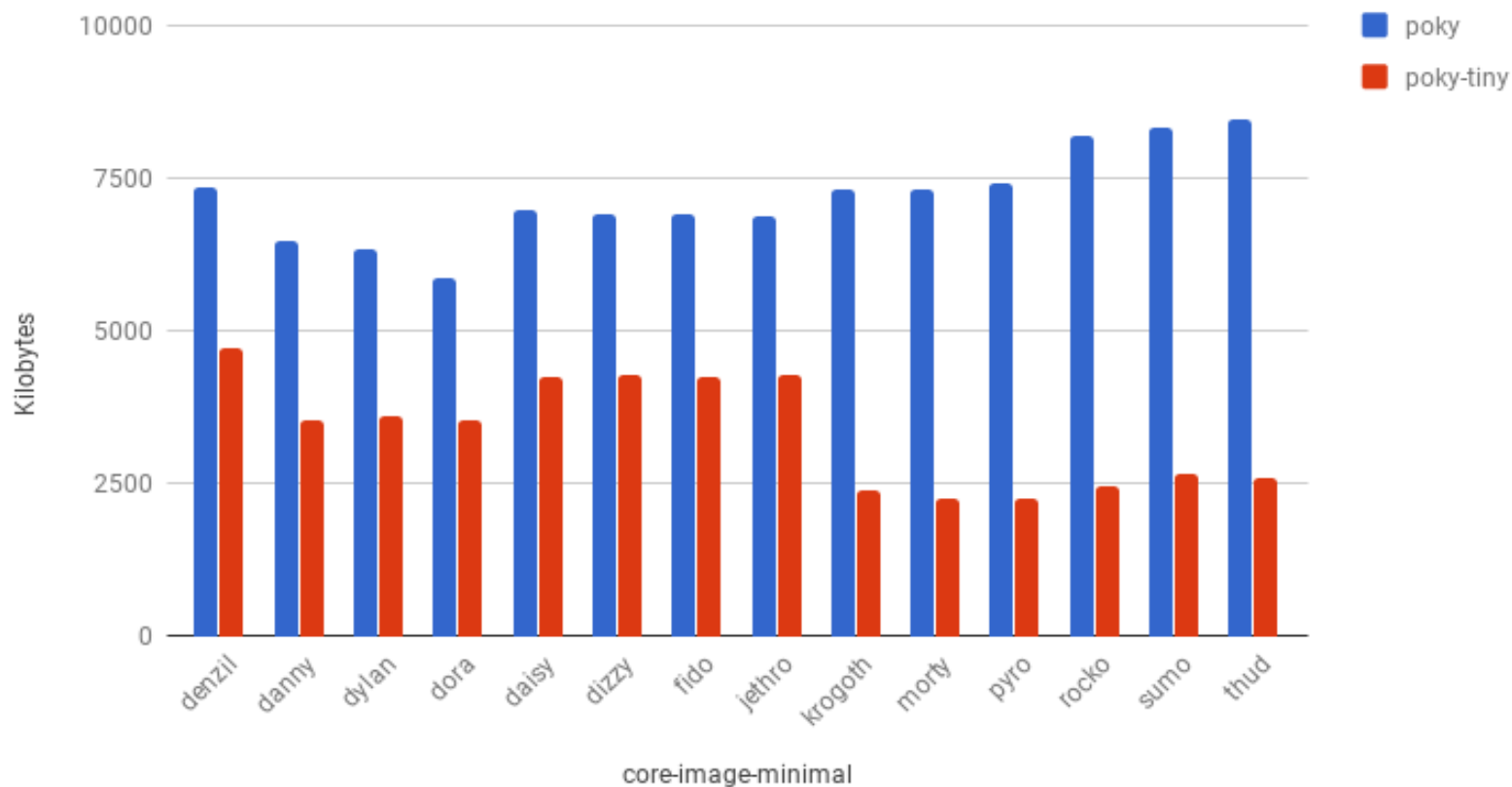
- Added in Yocto Project denzil release in April 2012
- “Poky-tiny is intended to define a tiny Linux system comprised of a Linux kernel tailored to support each specific MACHINE and busybox.” (from poky-tiny.conf)
- As with poky, intended to act as a starting point for your own distribution.
- Caveats:
 - Only builds for qemux86 (and recently qemux86-64) by default
 - Only supports core-image-minimal
 - So only has a barebones kernel, libc, and BusyBox
- Documented (somewhat) at:
 - <https://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#building-a-tiny-system>

poky-tiny contents

```
$ cat installed-package-sizes.txt
606 KiB musl
548 KiB busybox
23    KiB netbase
4     KiB update-alternatives-opkg
3     KiB busybox-udhcpc
3     KiB busybox-mdev
3     KiB base-files
2     KiB run-postinsts
2     KiB busybox-syslog
0     KiB packagegroup-core-boot
0     KiB base-passwd
```

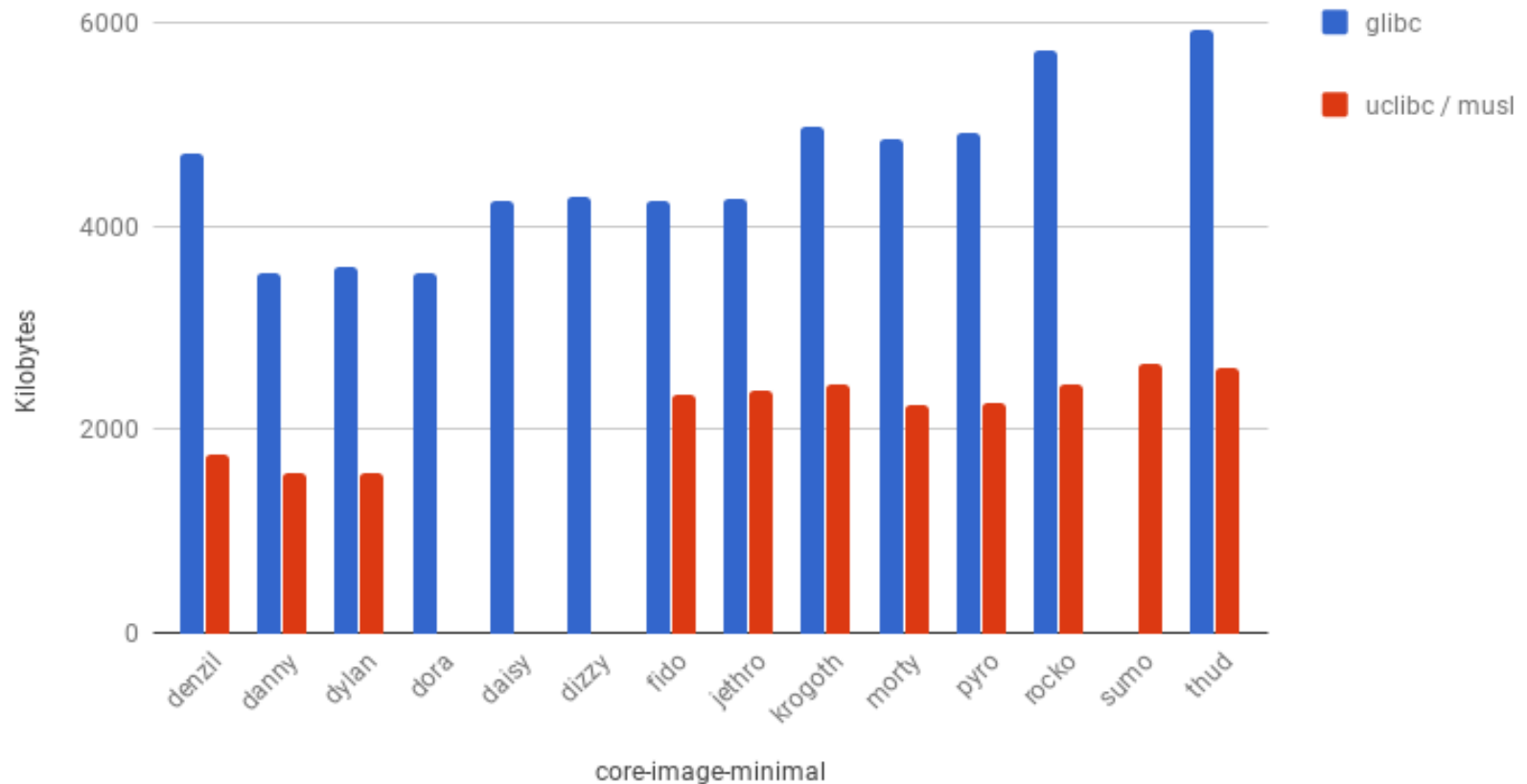
So how big is poky-tiny?

poky vs poky-tiny (defaults, qemu86)



So how big is poky-tiny? (continued)

poky-tiny (glibc vs uclibc/musl, qemu86)



So how big is poky-tiny? (Notes / Observations)

- Numbers pulled from data generated by buildhistory class
- Missing uclibc sizes for dora - dizzy releases due to build issues
- Noticeable trend of glibc increasing in size over time
- No obvious dramatic size difference between uclibc and musl
- Installed package size != root filesystem size
 - There is some overhead due to the inodes from the many small files used by update-alternatives-opkg metadata
 - The root filesystem is assembled from binary packages, and the update-alternatives-opkg tool is used to handle alternate providers of binaries (e.g. BusyBox versus util-linux)
 - `Removed in read-only images or with FORCE_RO_REMOVE option`

poky-tiny changes versus poky

- TCLIBC
- ENABLE_WIDEC
- USE_NLS
- DISTRO_FEATURES
- linux-yocto-tiny

poky-tiny changes: TCLIBC

- TCLIBC variable selects the standard C library to use; default value is glibc, as of the krogoth release the other option is musl (previously was uclibc)
- musl (<https://wiki.musl-libc.org/>) is a lightweight C library implementation
 - Actively maintained, MIT licensed
 - In addition to binary size benefits, there are significant runtime memory usage ones as well
- See http://www.etalabs.net/compare_libcs.html for a detailed comparison
- While the recipes in oe-core are test built with musl, recipes from other layers may not work out of the box, and other software may require patching to build
 - Almost all the recipes in meta-openembedded build against musl, the remaining handful are actively being worked on
 - Typical failures are due to accidentally relying on non-standard glibc extension or definitions

poky-tiny changes versus poky

- TCLIBC
- ENABLE_WIDEC
- USE_NLS
- DISTRO_FEATURES
- linux-yocto-tiny

poky-tiny changes: TCLIBC

- TCLIBC variable selects the standard C library to use; default value is glibc, as of the krogoth release the other option is musl (previously was uclibc)
- musl (<https://wiki.musl-libc.org/>) is a lightweight C library implementation
 - Actively maintained, MIT licensed
 - In addition to binary size benefits, there are significant runtime memory usage ones as well
- See http://www.etalabs.net/compare_libcs.html for a detailed comparison
- While the recipes in oe-core are test built with musl, recipes from other layers may not work out of the box, and other software may require patching to build
 - Almost all the recipes in meta-openembedded build against musl, the remaining handful are actively being worked on
 - Typical failures are due to accidentally relying on non-standard glibc extension or definitions

poky-tiny changes: `ENABLE_WIDEC`

- `ENABLE_WIDEC` variable controls wide character support for the ncurses terminal library
- Disabling ncurses wide character support only affects console applications
- However, not all applications will build with it disabled
 - `core-image-minimal` and `core-image-full-cmdline` images build, `core-image-sato` does not
- Size savings are not necessarily dramatic
 - About 200 KB if the image pulls in ncurses
 - ATM `core-image-minimal` does not actually pull in any packages that have ncurses as a dependency...

poky-tiny changes: `USE_NLS`

- `USE_NLS` variable controls native language support for applications, i.e. internationalization via the gettext library
- Disabling NLS might be problematic if you have applications using gettext to provide internationalized output
- However, again not all applications will build with it disabled
 - core-image-minimal and core-image-full-cmdline build, core-image-sato does not
- Size savings are dependent on application usage of gettext
 - No savings in core-image-minimal, about 2 MB in core-image-full-cmdline

poky-tiny changes: DISTRO_FEATURES

- DISTRO_FEATURES variable controls software feature support
 - Mostly translates to configure script options, but some features add kernel module and runtime package dependencies
- poky-tiny removes almost all of the default features that poky enables, leaving on IPv4 and IPv6 support on as well as a couple of other base features
- The features you need are largely dependent on your target image contents
 - e.g. x11, pulseaudio, many fine-grained libc features for glibc
- See Chapter 14 of the Yocto Project Reference Manual for a breakdown of DISTRO and MACHINE features
 - <https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#ref-features>

poky-tiny changes: linux-yocto-tiny

- Provides a highly pruned kernel configuration
- qemu86 specific
- Needs to be over-ridden with `PREFERRED_PROVIDER_virtual/kernel` if you want to test poky-tiny on another architecture
- General kernel size reduction guidelines apply, i.e. only enable features and drivers for the target platform
- A static kernel without modules is a win if possible since it results in an overall size reduction

Common image feature / package sizes

- Note that size numbers include all the dependencies that are pulled in
- Package management
 - rpm: ~102 MB (includes OpenSSL, Python 3, etc.)
 - deb: ~22 MB
 - ipk: ~4 MB
- SSH daemon:
 - OpenSSH (and OpenSSL): ~6.7 MB
 - Dropbear: ~300 KB
- Systemd: ~30 MB
- Python 2.7: ~4 MB, ~40 MB with all standard modules
- Python 3.5: ~17 MB, ~64 MB with all standard modules

core-image-minimal-initramfs

- Another option is to build the target image into an initramfs and attach it to the kernel
 - initramfs ends up as cpio.gz attached to the kernel image
 - unpacked into a memory filesystem and used as the rootfs
- Makes for a smaller overall combined image size at the expense of memory usage
- core-image-minimal-initramfs is probably best viewed as a template, copy and prune out anything not needed for your target

Other size reduction options

- Splitting files out of a package with `FILES_${PN}-foo`
 - Example use would be to pick out a tool from `core-utils`, since it is not split into per-tool packages like `util-linux`
- A more extreme example is removing all `.py` Python source files, leaving only the compiled `.pyc` files
 - Currently requires modifying `distutils-common-base.bbclass` to make it generic for all Python modules
- Use `ROOTFS_POSTPROCESS_COMMAND` to remove files from the image
 - e.g. removing unneeded `update-alternatives-opkg` metadata
 - https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#var-ROOTFS_POSTPROCESS_COMMAND
- If using `glibc`, tweak `IMAGE_LINGUAS` to remove unwanted locales

Summary / Recommendations

- If starting out, take poky-tiny.conf as a starting point to define your own distribution configuration, then add things to it
- Otherwise, it is likely that switching to using musl will provide the biggest immediate improvement
- Use the buildhistory class to help simplify investigating what is taking up space in your image
 - <https://www.yoctoproject.org/docs/current/ref-manual/ref-manual.html#maintaining-build-output-quality>
- For kernel (and BusyBox) size reduction start with the guidelines at:
 - <https://www.yoctoproject.org/docs/current/dev-manual/dev-manual.html#trim-the-kernel>



Activity Nine

(Fun with) Libraries/SDK and OE/YP
Robert Berger

(Fun with) Libraries/SDK and OE/YP

- **Slides are available at:**
 - <https://wiki.yoctoproject.org/wiki/File:Having-fun-handouts.pdf>
- **Home “Ship It” content download:**
 - <http://www.rlbl.me/ypdevd2018/>
- **Master DevDay Slides Page:**
 - https://wiki.yoctoproject.org/wiki/DevDay_Edinburgh_2018



Activity Ten

Yocto Project - Rarely asked questions

Khem Raj

How to add layers to Workspace

- **bitbake-layers**
 - add-layer/remove-layer – Add/Remove a layer to workspace
 - show-layer – Show current list of used layers
 - show-recipes – List available recipes
 - show-appends – List appends and corresponding recipe
 - show-overlayed – List overlayed recipes

Are there some Workspace helper Tools

- **bitbake-whatchanged**
 - print what will be done between the current and last builds

```
$ bitbake core-image-sato
# Edit the recipes
$ bitbake-whatchanged core-image-sato
```

How to make changes in workspace

- **Prepare a package to make changes**
`# devtool modify <recipe>`
- **Change sources**
 - Change into workspace/sources/<recipe>
 - Edit
- **Build Changes**
`$ devtool build <recipe>`
- **Test changes**
`$ devtool deploy-target <recipe> <target-IP>`
- **Make changes final**
`$ devtool finish <recipe> <layer>`

How to enquire package information ?

- `oe-pkgdata-util` - queries the `pkgdata` files written out during `do_package`

subcommands:

<code>lookup-pkg</code> and	Translate between recipe-space package names runtime package names
<code>list-pkgs</code>	List packages
<code>list-pkg-files</code>	List files within a package
<code>lookup-recipe</code>	Find recipe producing one or more packages
<code>package-info</code> one or	Show version, recipe and size information for more packages
<code>find-path</code>	Find package providing a target path
<code>read-value</code> packages	Read any <code>pkgdata</code> value for one or more
<code>glob</code>	Expand package name glob expression

Use `oe-pkgdata-util <subcommand> --help` to get help on a specific command

How to run meta-data self tests (unit tests)

- **oe-selftest**

Script that runs unit tests against bitbake and other Yocto related tools. The goal is to validate tools functionality and metadata integrity

- List available tests

\$ **oe-selftest -l**

- Run all tests

\$ **oe-selftest --run-all-tests**

- Run Selective Unit Test

\$ **oe-selftest -r devtool
devtool.DevtoolTests.test_devtool_add_fetch_simple**

- <https://wiki.yoctoproject.org/wiki/Oe-selftest>

How to run image auto-test

- Can test image (**-c testimage**) (**-c testimage_auto**)

```
INHERIT += "testimage"  
DISTRO_FEATURES_append = " ptest"  
EXTRA_IMAGE_FEATURES_append = " ptest-pkgs"  
##TEST_SUITES = "auto"  
TEST_IMAGE_qemuall = "1"  
TEST_TARGET_qemuall = "qemu"  
TEST_TARGET ?= "simpleremote"  
TEST_SERVER_IP = "10.0.0.10"  
TEST_TARGET_IP ?= "192.168.7.2"
```

- Testing SDK (**-c testsdk** and **-c testsdkext**)

```
INHERIT += "testsdk"  
SDK_EXT_TYPE = "minimal"
```

- <https://www.yoctoproject.org/docs/latest/dev-manual/dev-manual.html#performing-automated-runtime-testing>

How to send Code upstream

- **create-pull-request**

Examples:

```
create-pull-request -u contrib -b joe/topic
```

- **send-pull-request**

Examples:

```
Send-pull-request -a -p pull-XXXX
```

How to Customize Distro

- Example poky-lsb

```
require conf/distro/poky.conf
require conf/distro/include/security_flags.inc
```

```
DISTRO = "poky-lsb"
DISTROOVERRIDES = "poky:linuxstdbase"
```

```
DISTRO_FEATURES_append = " pam largefile opengl"
PREFERRED_PROVIDER_virtual/libx11 = "libx11"
```

```
# Ensure the kernel nfs server is enabled
KERNEL_FEATURES_append_pn-linux-yocto = " features/nfsd/nfsd-
enable.scc"
```

```
# Use the LTSI Kernel for LSB Testing
PREFERRED_VERSION_linux-yocto_linuxstdbase ?= "4.14%"
```

How to Customize Machine

- **odroid-c2.conf**

```
#@TYPE: Machine
#@NAME: odroid-c2
#@DESCRIPTION: Machine configuration for
odroid-c2 systems
#@MAINTAINER: Armin Kuster
<akuster808@gmail.com>

require conf/machine/include/amlogic-
meson64.inc

DEFAULTTUNE ?= "aarch64"
include conf/machine/include/odroid-
default-settings.inc

EXTRA_IMAGEDEPENDS += "u-boot secure-
odroid"

KERNEL_DEVICETREE_FN = "meson-gxbb-
odroidc2.dtb"

KERNEL_DEVICETREE = "amlogic/meson-gxbb-
odroidc2.dtb"
```

- **odroid-c2-hardkernel.conf**

```
#@TYPE: Machine
#@NAME: odroid-c2-hardkernel
#@DESCRIPTION: Machine configuration for
odroid-c2 systems using uboot/kernel
from hardkernel supported vendor tree
#@MAINTAINER: Armin Kuster
<akuster808@gmail.com>

require conf/machine/odroid-c2.conf

SERIAL_CONSOLE = "115200 ttyS0"
UBOOT_CONSOLE = "console=ttyS0,115200"

KERNEL_DEVICETREE_FN_odroid-c2-
hardkernel = "meson64_odroidc2.dtb"
KERNEL_DEVICETREE_odroid-c2-hardkernel =
"meson64_odroidc2.dtb"
```

How to setup/use feeds ?

- **Configuring feeds in image**

```
$ PACKAGE_FEED_URI = "http://10.0.0.10:8000/"
```

- **Start a http server in deploydir**

```
$ cd tmp/deploy/ipk
```

```
$ python3 -m http.server 8000
```

- **Run Package manager on booted target**

```
$ opkg update
```

```
$ opkg upgrade
```

Questions



Activity Eleven

Tools, Toaster, User Experience

David Reyna

Toaster: Latest Features (1/2)

- **Toaster Documentation**
 - <https://www.yoctoproject.org/docs/latest/toaster-manual/toaster-manual.html>
- **Toaster Service Without a Web Server (“noweb”)**
 - Good for capturing command line build(s) directly into the db
- **Toaster Service Without Remote Builds (“nobuild”)**
 - Good for sharing build local status, without enabling external people creating projects and starting builds on your host
- **Toaster Service – Build Status within Containers**
 - New REST/JSON API to access the progress and health of bitbake builds via HTTP; very handy for containers
 - Build Status options: “Completed”, “In Progress”, “Specific Status”

Toaster: Latest Features (2/2)

- **Compatibility between Command Line and Toaster builds**
 - New “Import command line build” option
 - New “Merge Toaster Settings” into standard conf files”

Create a new project

Project name (required)

Project type:

☒ New project

☐ Import command line project

Release 

Yocto Project master 

Toaster will run your builds using the tip of the [Yocto Project Master branch](#).

☐ Merged Toaster settings (Command line user compatibility) 

[Create project](#) To create a project, you need to enter a project name

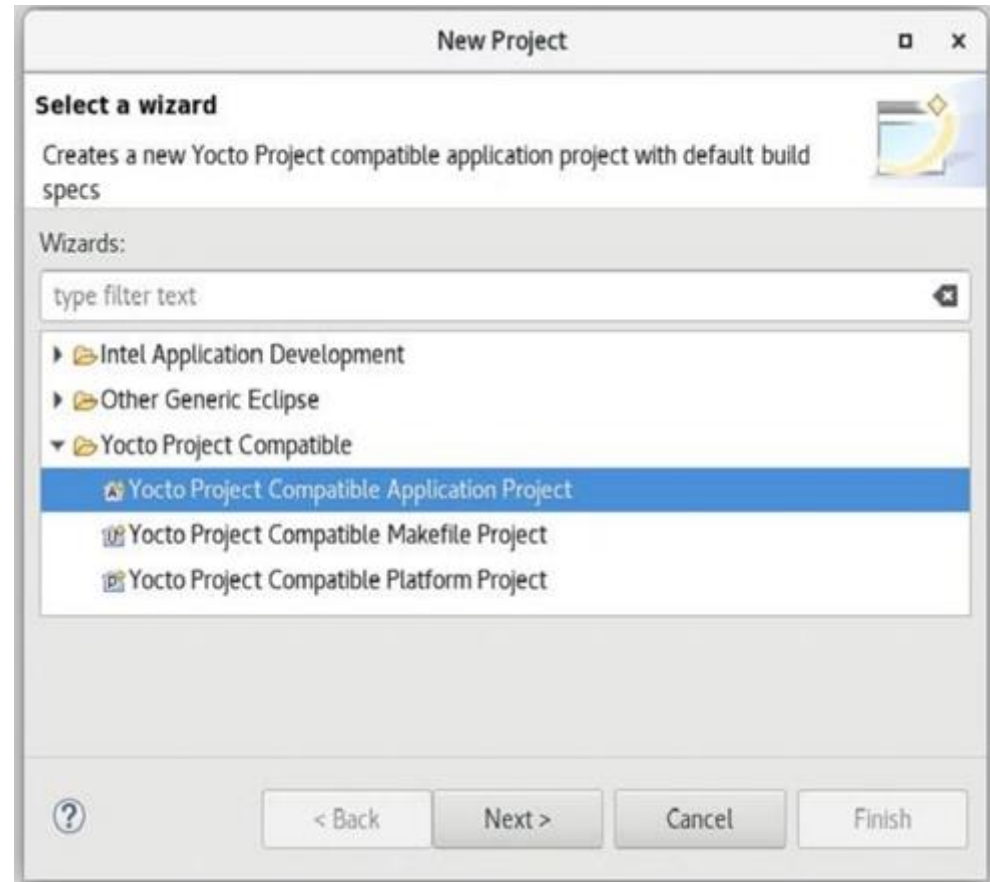
Intel System Studio 2019: Yocto Project Compatible

- The Wind River Application and Project plug-ins have been shared with Intel System Studio, with the idea of open sourcing them to Eclipse.org
- Implementation is architecture agnostic
- **Application Project Features:**
 - Awareness of YP compatible SDKs/eSDKs
 - Ability to register multiple SDKs
 - Automatic generation of “Build Specs” for each machine variant in each SDK
 - Ability to enable/disable debug flags
 - Debugger deploy and access over GDB/TCF
 - Set of sample applications

Intel System Studio 2019: Yocto Project Compatible

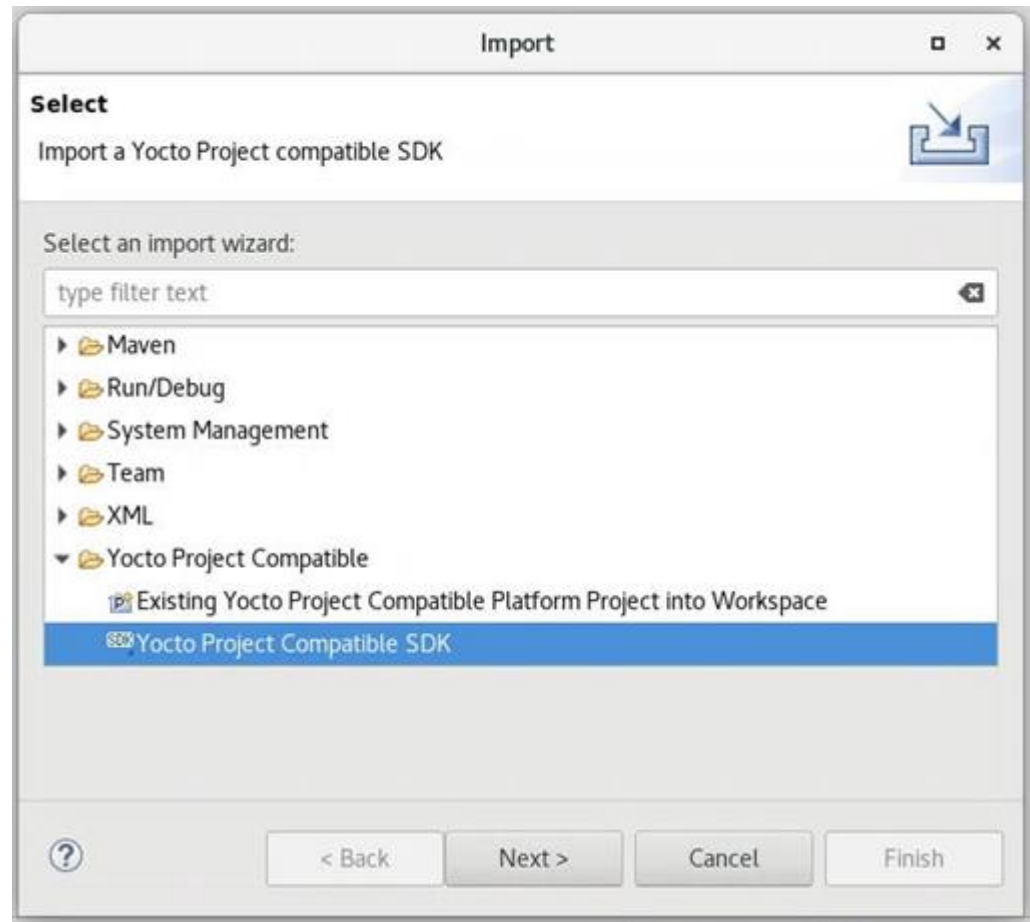
- **Platform Project Features:**

- Configuration/Updates via Toaster
- Basic build targets directly from ISS
- Eclipse-based Kernel Configuration Tool
- Tree view to browse deploy artifacts



Intel System Studio 2019: Yocto Project Compatible

- **Import:**
 - Existing command line project
 - Existing SDK/eSDK



New Security Response Tool (SRTool)

- *While there is heighten awareness about device vulnerabilities, what is often missing is awareness about the process of managing the security response process itself*
- **Wind River is sharing to open source a tool to help manager the organization's security response management:**
 - Better ways to handle 1000+ CVEs per month
 - Better ways to connect CVE's to defects to product
 - Better ways to allow easy access to the full vulnerability status, generate reports, clean exports to public CVE DB
 - Better ways to use automation to keep all the data sources automatically up to date
- **Community Page:**
 - https://wiki.yoctoproject.org/wiki/Contribute_to_SRTool
- **ELCE Presentation:**
 - <https://sched.co/HOLr>



Activity Nine

A User's Experience Henry Bruce

What I'll be talking about

- **Learnings from my painful ramp on Yocto**
- **Get similar experiences from the audience**
- **Funnel these learnings into topics in the new Development Tasks Manual**
- **Review improvements in usability over the past few years**

General areas I'll be covering

- **Proxies**
- **Debugging build errors**
- **Writing recipes**
- **Recipes vs. Packages**
- **Application Development**
- **Cool things I stumbled across**
- **Improvements**

Some context

- **Started as an open source neophyte**
 - Had never really used git or dug into Linux
- **Spent six months in extreme pain**
 - Mainly due to OpenJDK
- **For the next year I was learning**
- **After 2 years I felt I could competently help others**
- **Over 3 years later, there's still so much to learn**
- **I should have taken better notes**

Proxies

- **A common problem for new users**
- **Proxy wiki page has 135k hits**
 - https://wiki.yoctoproject.org/wiki/Working_Behind_a_Network_Proxy
- **Environment variable approach covers most cases**
 - But fails when non-fetch tasks reach out to network
 - This includes most node.js recipes
 - How important is network isolation for post fetch tasks?
- **Chameleonsocks has been failsafe for me**
 - But some say this an abuse of docker
- **What's your solution?**

When things go wrong

- **You've gone through the quick start guide and have figured out how to add packages to an image**
- **You're feeling pretty good but then you get a build error.**
- **Due to many moving parts it's easy to panic when something breaks**
 - Or at least it was for me

It broke – what would have helped?

- **Nicer output from bitbake on bad directory/file names**
- **Understanding the task pipeline**
 - fetch / unpack / configure / build / install / package
- **Knowing how to generate dependency graph**
- **Decoding “magic” folder names in tmp/work**
- **Understanding recipe vs. package**
- **Knowing how to run specific task for specific recipe**
- **Knowing what’s packaged and in rootfs**

Recipes

- **Plenty of resources to writing simple recipes**
 - But then there seems to be a gap
- **Can be hard to work out what a recipe is doing**

```
pn = d.getVar('PN', 1)
metapkg = pn + '-dev'
d.setVar('ALLOW_EMPTY_' + metapkg, "1")
blacklist = [ metapkg ]
metapkg_rdepends = [ ]
packages = d.getVar('PACKAGES', 1).split()
for pkg in packages[1:]:
    if not pkg in blacklist and pkg.endswith('-dev'):
        metapkg_rdepends.append(pkg)
d.setVar('RRECOMMENDS_' + metapkg, ' '.join(metapkg_rdepends))
```

- **Walk through a couple of good citizens in oe-core?**

Recipes and packages

- **Easy to assume there is 1:1 mapping**
- **Sometimes there isn't**
 - devtool search rocks
- **Sub-packages can trip you up**
 - OpenCV vs. UPM
- **Creating sub-packages for large project seems to be the “right” pattern**
 - But I can't find obvious guidance in docs
- **Thoughts?**

Application Development

- **I was initially confused by the terminology**
 - ADT, SDK, eSDK, toolchain
- **In retrospect ADT seemed the clearest naming**
 - I'm now working on a real-time SDK
 - Yocto built Linux is our initial target platform
 - I tell my team to develop for the target using the Yocto SDK
 - Confusion all round
- **Eclipse**
 - Broken when I first tried
 - I need to get back to it

Improvements

- **eSDK and devtool**
- **Recipetool**
 - ROS support
 - Is it worth investing more, or do returns diminish?
- **Package feeds**
 - Credit to dnf (setting server means build checks if it's there)
 - But package-index is a big gotcha
- **Development Tasks Manual**
- **CROPS**
 - Who's using it?

Cool things I stumbled across

- **PACKAGECONFIG**
- **INSANE_SKIP**
- **Overrides**
- **Layer dependencies**
- **Setting package variables from outside recipe**
- **Conditional logic with python**
 - Adding package to image if its layer is present
- **What's your favorite?**



Questions and Answers

A decorative pattern of overlapping hexagons in various shades of gray, located in the top-left corner of the slide.

Thank you for your participation!

