Testing mainline Linux kernel and bootloader

OpenEmbedded Workshop 2025

Michael Opdenacker

Root Commit

February 3, 2025

Michael Opdenacker

Embedded Linux consultant and trainer

- https://rootcommit.com/about/michael-opdenacker/
- Former founder of Bootlin
- New founder of Root Commit
- Offering embedded Linux training courses with a focus on practical activities, interactivity and learning techniques. https://rootcommit.com/training/
- Free Software enthusiast and advocate (member of April.org)

Learn by doing

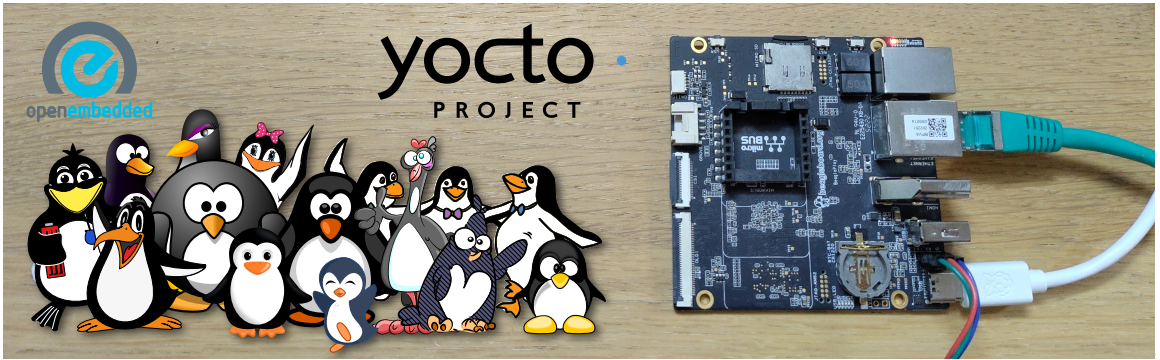
Yocto Project & OpenEmbedded training course

https://rootcommit.com/training/yocto/

root commit

In person or online course

- 24 hours (online) / 4 days (in-person)
- 75 % of practical activities
- Online: do labs by yourself instead of watching demos
- No exhaustive theory
- Challenging, varied and fun learning techniques
- Use your favorite Linux distro
- Limited to 8 participants
- Online: video recordings of lectures



Introduction

Goals of this presentation

root commit

- Official BSP layers usually support a specific kernel and bootloader version.
- Show how to override the default layers to run the kernel and U-Boot version of your choice, especially the latest development branch or stable releases, while testing with exhaustive userspace stacks.
- This should help to contribute to the upstream kernel and bootloader projects, and get your system ready for future releases of the Yocto Project.

4 example machines

- Officially supported by Poky:
  - `genericarm64`
  - `beaglebone-yocto`
- Supported by third-party layers:
  - `beagleplay-ti`
  - `raspberrypi5`

Standard builds

genericarm64 — building

Great for people who don't have the hardware!

```
$ mkdir -p ~/oe/genericarm64
$ cd ~/oe/genericarm64
$ git clone -b styhead \
https://git.yoctoproject.org/poky
$ cd poky
$ source oe-init-build-env
```

Then set `MACHINE` in `conf/local.conf`

```
MACHINE = "genericarm64"
```

Also add a bootloader to the image, so that we can boot it with QEMU.

```
IMAGE_INSTALL:append = " u-boot"
```

Then generate an image:

```
$ bitbake core-image-minimal
```

genericarm64 — testing

Boot the image through QEMU

```
$ runqemu slirp nographic
```

See the machine booting and the U-Boot version:

```
...
U-Boot 2024.07 (Jul 01 2024 - 18:07:18 +0000)
...
```

Log in as `root` (no password) and check the Linux kernel version:

```
root@genericarm64:~# uname -r
6.6.69-yocto-standard
```

To exit QEMU: `[Ctrl] a x`.

beaglebone-yocto — building

- Primarily for the Beaglebone Black board, but other 32 bit Beaglebones (TI AM335x) should work too.
- The only hardware board officially supported by Yocto (other boards supported through third-party layers)

```
$ mkdir -p ~/oe/beaglebone-yocto
$ cd ~/oe/genericarm64
$ git clone -b styhead \
https://git.yoctoproject.org/poky
$ cd poky
$ source oe-init-build-env
```

Then set `MACHINE` in `conf/local.conf`

```
MACHINE = "beaglebone-yocto"
```

Then generate an image:

```
$ bitbake core-image-minimal
```

And flash the image on a micro-SD card (let's assume it's `/dev/mmcblk0`:

```
$ sudo bmaptool copy \
 tmp/deploy/images/beaglebone-yocto/core-
image-minimal-beaglebone-yocto.rootfs.wic \
 /dev/mmcblk0
```

beaglebone-yocto — testing

root commit

Connect the board UART to your PC (assume it's `/dev/ttyUSB`), and start a terminal emulator to get serial console messages:

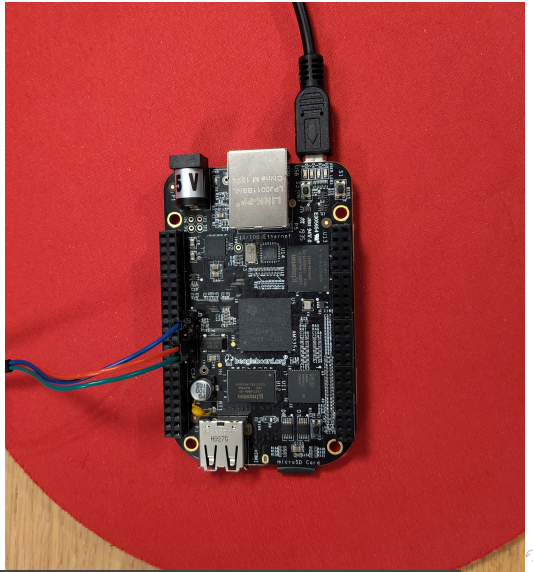
```
$ tio /dev/ttyUSB0
```

Insert the micro-SD card, hold the `USR` button, connect the power cable, release `USR`. See the machine booting and the U-Boot version:

```
...
U-Boot 2024.07 (Jul 01 2024 - 18:07:18 +0000)
...
```

Log in as `root` and check the Linux kernel version:

```
root@beaglebone-yocto:~# uname -r
6.6.21-yocto-standard
```



beagleplay-ti — building

root commit

- This time the BeaglePlay board is supported through a third-party layer (`meta-ti`), which also depends on `meta-arm` depending on `meta-arm-toolchain`.
- We need to use the Scarthgap LTS version, as there are currently issues with `master` and `meta-ti` doesn't support Styhead.

```
$ mkdir -p ~/oe/beagleplay-ti
$ cd ~/oe/beagleplay-ti
$ git clone -b scarthgap \
    https://git.yoctoproject.org/poky
$ git clone -b scarthgap \
    https://git.yoctoproject.org/git/meta-arm
$ git clone -b scarthgap \
    https://git.yoctoproject.org/git/meta-ti
$ cd poky
$ source oe-init-build-env
```

beagleplay-ti — building

Then add the necessary layers:

```
$ bitbake-layers add-layer \
../../meta-arm/meta-arm-toolchain
$ bitbake-layers add-layer \
../../meta-arm/meta-arm
$ bitbake-layers add-layer \
../../meta-ti/meta-ti-bsp
```

Then set `MACHINE` in `conf/local.conf`

```
MACHINE = "beagleplay-ti"
```

Then generate an image:

```
$ bitbake core-image-minimal
```

And flash the image on a micro-SD card:

```
$ sudo bmaptool copy \
 deploy-ti/images/beagleplay-ti/core-image-minimal-
beagleplay-ti.rootfs.wic.xz
```

beagleplay-ti — testing

root commit

Connect the board UART to your PC and start a terminal emulator:

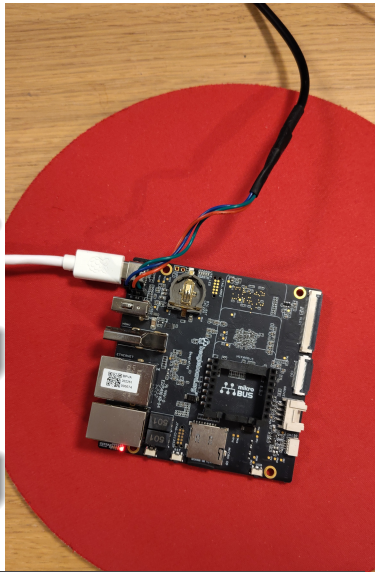
```
$ tio /dev/ttyUSB0
```

Insert the micro-SD card, hold the `USR` button, connect the power cable, release `USR`. See the machine booting and the U-Boot version:

```
...
U-Boot 2024.04-ti-g29d0c23d67ee (Nov 29 2024 - 11:41:54 +0000)
...
```

Log in as `root` and check the Linux kernel version:

```
root@beagleplay-ti:~# uname -r
6.6.58-ti-01497-ga7758da17c28-dirty
```



raspberrypi5 — building

- This time the `raspberrypi5` machine just needs the `meta-raspberrypi` layer.
- Using the `master` branch of this layer as it supports both Styhead and Walnascar

```
$ mkdir -p ~/oe/raspberrypi5
$ cd ~/oe/raspberrypi5
$ git clone -b styhead \
    https://git.yoctoproject.org/poky
$ git clone \
    git://git.yoctoproject.org/meta-raspberrypi
$ cd poky
$ source oe-init-build-env
```

Add your layer:

```
$ bitbake-layers add-layer \
   ../../meta-raspberrypi
```

Then set `MACHINE` in `conf/local.conf`

```
MACHINE = "raspberrypi5"
```

Then generate an image:

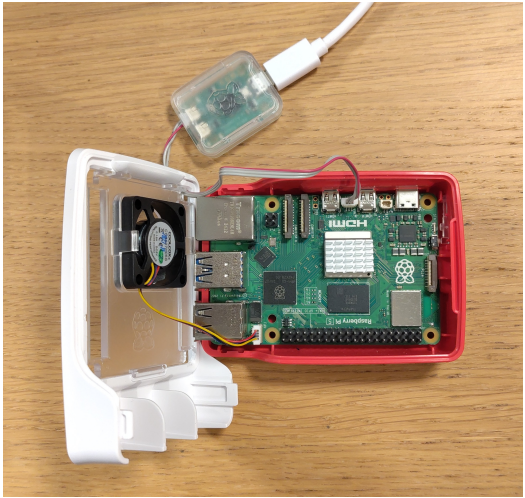
```
$ bitbake core-image-minimal
```

Connect the Raspberry Pi Debug Probe to your board and PC, and start a terminal emulator:

```
$ tio /dev/ttyACM0
```

Insert the micro-SD card, and power up the board. Raspberry Pi uses a custom bootloader instead of U-Boot. Log in as `root` and check the Linux kernel version:

```
root@raspberrypi5:~# uname -r
6.6.63-v8-16k
```



Building standard image: main takeways

root commit

With OpenEmbedded / Yocto Project
it's very easy to generate a reference image:

- Clone build system sources
- Define `MACHINE`
- If necessary, add BSP layers supporting `MACHINE`
- Generate, flash and boot the image

Builds with the mainline Linux kernel

Builds with the mainline Linux kernel

genericarm64

Custom layer

Let's create a custom layer to override the reference ones.

```
$ cd ~/oe/
$ bitbake-layers create-layer meta-mainline
$ bitbake-layers add-layer meta-mainline
```

genericarm64: find the kernel recipe

root commit

Let's look for the available kernel recipes:

```
$ bitbake-layers show-recipes "linux*"
=== Matching recipes: ===
linux-dummy:
  meta                 unknown (skipped: PREFERRED_PROVIDER_virtual/kernel set to linux-yocto, not linux-dummy)
linux-firmware:
  meta                 1:20240909
linux-libc-headers:
  meta                 6.10
linux-yocto:
  meta                 6.6.69+git
  meta                 6.10 (skipped: incompatible with machine genericarm64 (not in COMPATIBLE_MACHINE))
  meta                 6.10 (skipped: incompatible with machine genericarm64 (not in COMPATIBLE_MACHINE))
linux-yocto-dev:
  meta                 unknown (skipped: Set PREFERRED_PROVIDER_virtual/kernel to linux-yocto-dev to enable it)
linux-yocto-rt:
  meta                 6.10 (skipped: incompatible with machine genericarm64 (not in COMPATIBLE_MACHINE))
  meta                 6.6 (skipped: incompatible with machine genericarm64 (not in COMPATIBLE_MACHINE))
linux-yocto-tiny:
  meta                 6.10 (skipped: incompatible with machine genericarm64 (not in COMPATIBLE_MACHINE))
  meta                 6.6 (skipped: incompatible with machine genericarm64 (not in COMPATIBLE_MACHINE))
```

→`linux-yocto` is the recipe we're looking for (not skipped), in the `meta` layer. It turns out to be under `meta/recipes-kernel/`.

Find mainline Linux commit

root commit

Let's find the commit id for Linux 6.13:

```
$ git clone https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
$ cd linux
$ git show v6.13 | grep commit
commit ffd294d346d185b70e28b1a28abe367bbfe53c04
```

As you can see, recipes using Git sources need to set the `SRCREV` variable to such a commit id.

Custom kernel recipe

root commit

Let's create a custom recipe for the mainline Linux sources. We will store it in `recipes-kernel` as in the original recipes.

```
$ cd ~/oe/
$ cd linux-mainline
$ mkdir -p recipes-kernel/linux-mainline
```

Custom kernel recipe

Let's create the `linux-mainline_6.13.bb` recipe file itself:

```
require linux-mainline.inc
# Linux 6.13
SRCREV = "ffd294d346d185b70e28b1a28abe367bbfe53c04"
```

And the associated `linux-mainline.inc` file:

```
DESCRIPTION = "Mainline Linux kernel"
LICENSE = "GPL-2.0-only"
LIC_FILES_CHKSUM = "file://COPYING;md5=6bc538ed5bd9a7fc9398086aedcd7e46"
inherit kernel
S = "${WORKDIR}/git"
SRC_URI = "git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git;branch=master;protocol=https \
           file://defconfig"
PROVIDES = "virtual/kernel"
```

This file can be reused for multiple kernel versions.

Kernel configuration for genericarm64

We also need to supply this `defconfig` file carrying the kernel configuration.
Go to the Linux sources directory and set:

```
$ export ARCH=arm64
$ sudo apt install gcc-aarch64-linux-gnu # cross toolchain
$ export CROSS_COMPILE=aarch64-linux-gnu- # matching toolchain
$ make defconfig # default configuration for all arm64 boards
$ make menuconfig
```

Here in `menuconfig`, let's trim the configuration

- In `Platform selection`, remove all supported SoCs
  (QEMU runs a "virt" machine)
- Look for `DRM`, and disable it.
  This removes a lot of unnecessary graphics and GPU code.

After exiting `menuconfig`, run `make savedefconfig`.

Store defconfig in meta-mainline

root commit

- We stored `defconfig` in a version specific and machine specific directory.
- See the documentation of `FILESPATH` for details about the search path for `file://` URIs.
- You can also run this command to see all directories between considered:

```
bitbake-getvar -r linux-mainline FILESPATH
```

```
~/oe/meta-mainline  tree .
.
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
├── recipes-example
│   └── example
│       └── example_0.1.bb
└── recipes-kernel
    └── linux-mainline
        ├── linux-mainline-6.13
        │   └── genericarm64
        │       └── defconfig
        ├── linux-mainline_6.13.bb
        └── linux-mainline.inc

8 directories, 7 files
```

genericarm64: update and boot image

Last thing, update `conf/local.conf`:

```
PREFERRED_PROVIDER_virtual/kernel = "linux-mainline"
```

Regenerate the image, boot it and check the kernel version:

```
root@genericarm64:~# uname -r
6.13.0
```

Builds with the mainline Linux kernel

beaglebone-yocto

Kernel configuration for beaglebone-yocto

We can reuse the same recipe for `beaglebone-yocto`.
All we need is a `defconfig` file:

```
$ export ARCH=arm
$ sudo apt install gcc-arm-linux-gnueabihf # cross toolchain
$ export CROSS_COMPILE=arm-linux-gnueabihf- # matching toolchain
$ make omap2plus_defconfig # config for TI OMAP2+ boards
$ make menuconfig
```

Simplify the configuration if you wish (like removing `DRM`, and then generate a `defconfig` file using `make savedefconfig`. After exiting `menuconfig`, run `make savedefconfig`.

Update image for beaglebone-yocto

root commit

- First thing to do — add your layer:

```
$ bitbake-layers add-layer ~/oe/meta-mainline
```

- Now save the new `defconfig` file under `linux-mainline-6.13/beaglebone-yocto/`.
- As for the previous board, update `conf/local.conf`:

```
PREFERRED_PROVIDER_virtual/kernel = "linux-mainline"
```

- Regenerate the image, flash it, and boot the board:

```
root@beaglebone-yocto:~# uname -r
6.13.0
```

~/oe/meta-mainline/recipes-kernel

```
.
└── linux-mainline
    ├── linux-mainline-6.13
    │   ├── beaglebone-yocto
    │   │   └── defconfig
    │   └── genericarm64
    │       └── defconfig
    ├── linux-mainline_6.13.bb
    └── linux-mainline.inc

5 directories, 4 files
```

Builds with the mainline Linux kernel

beagleplay-ti

Kernel configuration for beagleplay-ti

root commit

Let's create our kernel configuration file:

```
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-linux-gnu- # matching toolchain
$ make defconfig # default configuration for all arm64 boards
$ make menuconfig
```

Let's trim the configuration

- In `Platform selection`, remove all supported SoCs except TI K3.
- Disable `DRM`

Update image for beagleplay-ti

root commit

- First thing to do — add your layer:

```
$ bitbake-layers add-layer ~/oe/meta-mainline
```

- You will also have to modify `conf/layer.conf` to declare it as compatible with the `scarthgap` release:

```
LAYERSERIES_COMPAT_meta-mainline = "styhead scarthgap"
```

- Now save the new `defconfig` file under `linux-mainline-6.13/beagleplay-ti/`.
- As for the previous board, update `conf/local.conf`:

```
PREFERRED_PROVIDER_virtual/kernel = "linux-mainline"
```

- Regenerate the image, flash it, and boot the board:

```
root@beagleplay-ti:~# uname -r
6.13.0
```

~/oe/meta-mainline/recipes-kernel

```
.
└── linux-mainline
    ├── linux-mainline-6.13
    │   ├── beaglebone-yocto
    │   │   └── defconfig
    │   ├── beagleplay-ti
    │   │   └── defconfig
    │   └── genericarm64
    │       └── defconfig
    ├── linux-mainline_6.13.bb
    └── linux-mainline.inc
```

Builds with the mainline Linux kernel

raspberrypi5

Kernel configuration for raspberrypi5

root commit

Support for Raspberry Pi 5 is recent, introducted in Linux 6.12.
Let's create our kernel configuration file:

```
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-linux-gnu- # matching toolchain
$ make defconfig # default configuration for all arm64 boards
$ make menuconfig
```

Let's trim the configuration

- In `Platform selection`, remove all supported SoCs except `Broadcom SoC Support`.
- Disable `DRM`

Update image for raspberrypi5

- First thing to do — add your layer:

```
$ bitbake-layers add-layer ~/oe/meta-mainline
```

- Now save the new `defconfig` file under `linux-mainline-6.13/beagleplay-ti/`.
- As for the previous board, update `conf/local.conf`:

```
PREFERRED_PROVIDER_virtual/kernel = "linux-mainline"
```

- To avoid a build issue, as we're still using the `meta-raspberrypi` layer, you will also have to add this to `conf/local.conf`:

```
RPI_KERNEL_DEVICETREE_OVERLAYS = ""
```

- Regenerate the image, flash it, and boot the board:

```
root@raspberrypi5:~# uname -r
6.13.0
```

~/oe/meta-mainline/recipes-kernel/linux-mainline

```
.
├── linux-mainline-6.13
│   ├── beaglebone-yocto
│   │   └── defconfig
│   ├── beagleplay-ti
│   │   └── defconfig
│   ├── genericarm64
│   │   └── defconfig
│   └── raspberrypi5
│       └── defconfig
├── linux-mainline_6.13.bb
└── linux-mainline.inc
```

Using mainline kernel: main takeaways

root commit

You can use the standard features of the build system to replace the default kernel by the mainline one.

- Create a custom layer
- Add your own kernel recipe to this layer, inheriting `kernel.bbclass`.
- Set `SRCREV` to the commit id your want to use
- Add a `defconfig` file to `SRC_URI` for the configuration you wish to use.
- In `conf/local.conf`, use `PREFERRED_PROVIDER_virtual/kernel` to select your custom recipe to build the kernel.
- BitBakes makes it super easy to avoid duplication between machines: only configurations are machine specific.

meta-linux-mainline

A collection of Yocto Project recipes for the mainline and stable Linux kernel releases from kernel.org.

Setup information

Git repository

https://github.com/betafive/meta-linux-mainline.git web repo

Last commit: 4 days, 23 hours ago (master branch)

Maintainer

- Paul Barker email

Dependencies

The meta-linux-mainline layer depends upon:

- openembedded-core

Recipes | Updates

meta-linux-mainline recipes (7)

Export CSV | Search recipes

Recipe name	Version	Description
linux-mainline	6.13	Linux kernel (mainline)

Builds with the mainline U-Boot bootloader

Builds with the mainline U-Boot bootloader

genericarm64

New recipe for mainline U-Boot

root commit

- The U-Boot recipes are under `meta/recipes-bsp`. Let's create our own recipe under `recipes-bsp` too.
- Let's try to use version 2025.01.
- Create the `u-boot-mainline/u-boot-mainline_2025.01.bb` recipe, reusing code from the `u-boot_2024.07.bb` recipe →
- Here you see that we're also providing a configuration, but it has to be added as a patch, and specified through the `UBOOT_MACHINE` variable.
- We also need to add an `RPROVIDES`, as `core-image-minimal` `RDEPENDS` on `u-boot`.

```
require recipes-bsp/u-boot/u-boot-common.inc
require recipes-bsp/u-boot/u-boot.inc
PROVIDES = "virtual/bootloader"
RPROVIDES:${PN} = "u-boot"

# U-Boot 2025.01
SRCREV = "6d41f0a39d6423c8e57e92ebbe9f8c0333a63f72"
UBOOT_MACHINE = "custom_defconfig"
SRC_URI += "file://add-custom-defconfig.patch"

DEPENDS += "bc-native dtc-native gnutls-native python3-pyelftools-
native"
```

genericarm64: mainline U-Boot config

Clone the mainline U-Boot repository:

```
$ git clone https://source.denx.de/u-boot/u-boot.git
```

Create a branch to add a custom configuration:

```
$ git tag # find existing tags
$ git checkout -b genericarm64-custom-config v2025.01
```

Create our custom configuration:

```
$ make qemu_arm64_defconfig
$ make menuconfig
$ make savedefconfig
$ mv defconfig \
  configs/custom_defconfig
```

Create the patch:

```
$ git add configs/custom_defconfig
$ git commit -as "add custom defconfig"
$ git format-patch -1
```

Copy the patch to the recipe as `add-custom-defconfig.patch` in a machine-dependent subdirectory and modify this file to add patch status information after the `Signed-off-by` line:

```
Upstream-Status: Inappropriate
```

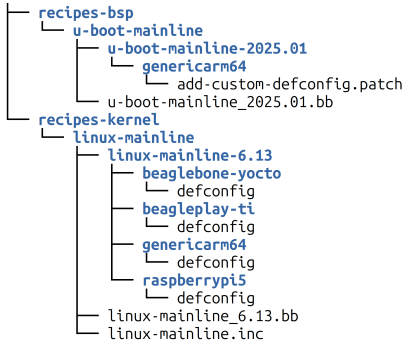
genericarm64: test mainline U-Boot

The last thing to do is add to `conf/local.conf`

```
PREFERRED_PROVIDER_virtual/bootloader = "u-boot-mainline"
```

Update the image, reflash it and boot it to check the U-Boot version:

```
U-Boot 2025.01 (Jan 07 2025 - 00:54:44 +0000)
```



Builds with the mainline U-Boot bootloader

beaglebone-yocto

beaglebone-yocto: mainline U-Boot config

The same procedure and settings work for `beaglebone-yocto`, as it is well supported in U-Boot too.

- Create a U-Boot configuration patch starting this time from `am335x_evm_defconfig`.
- Modify `conf/local.conf`:

```
PREFERRED_PROVIDER_virtual/bootloader = "u-boot-mainline"
```

- This works as expected:

```
U-Boot 2025.01 (Jan 07 2025 - 00:54:44 +0000)
```

```
recipes-bsp
└── u-boot-mainline
    ├── u-boot-mainline-2025.01
    │   ├── beaglebone-yocto
    │   │   └── add-custom-defconfig.patch
    │   └── genericarm64
    │       └── add-custom-defconfig.patch
    └── u-boot-mainline_2025.01.bb
recipes-kernel
└── linux-mainline
    ├── linux-mainline-6.13
    │   ├── beaglebone-yocto
    │   │   └── defconfig
    │   ├── beagleplay-ti
    │   │   └── defconfig
    │   ├── genericarm64
    │   │   └── defconfig
    │   └── raspberrypi5
    │       └── defconfig
    ├── linux-mainline_6.13.bb
    └── linux-mainline.inc
```

Mainline U-Boot on other boards

root commit

- `beagleplay-yocto`: the U-Boot recipe is much more complicated as U-Boot is built twice, once for the R5 core and one for the Cortex-A53. Booting mainline U-Boot on both requires more specific tweaking.
- `raspberrypi5`: it doesn't have U-Boot but its own bootloader instead!

Using mainline U-Boot: main takeaways

root commit

That's easy to do too for boards well supported in mainline:

- Add your own U-Boot recipe to your custom layer.
- There is no `u-boot.bbclass`, but you can `require` U-Boot recipe includes.
- Also figure out which `SRCREV` to use
- In our solution, U-Boot configuration provided as a patch.
- In `conf/local.conf`, also use `PREFERRED_PROVIDER_virtual/bootloader` to select your custom recipe to build the bootloader.
- BitBakes makes it super easy to avoid duplication between machines: only configurations are machine specific.

Questions? Comments?

- mo@rootcommit.com
- Signal: rootcommit.01
- XMPP: omichael@conversations.im

- Slides and sources available under the CC-By-SA 4.0 license
  `https://rootcommit.com/pub/conferences/2025/oe-workshop/yocto-mainline-linux-uboot/`
- Sources (LaTeX):
  `https://gitlab.com/rootcommit/yocto-mainline-kernel-bootloader/`