amarula solutions
Go Far, Go Together

TRAINING PROGRAMME

FIVE-DAY HANDS-ON TRAINING

Linux kernel and drivers

A guided tour from source code to device drivers

A five-day, hands-on course for embedded Linux engineers: kernel sources and configuration, booting, modules, Device Tree, the driver model, I2C and the driver frameworks, memory, interrupts, locking, DMA, debugging and upstream contribution.

DURATION	TRAINER	MINIMUM	PRICE
5 days	**Michael Trimarchi**	**3 participants**	**On request**

Delivered on site only. The course is not available online.

OVERVIEW

From the source tree to a working driver

The Linux kernel is the layer between your hardware and every user-space application. This five-day, hands-on course takes engineers from the kernel source tree to a working device driver: configuration and cross-compilation, booting with U-Boot, modules and the Device Tree, the device and driver model, I2C and the driver frameworks, and the kernel internals that drivers depend on — memory, VFS, MMIO, clocks, scheduling, interrupts, locking and DMA. The course closes with debugging and upstream contribution.

FORMAT	FOCUS	AUDIENCE	MINIMUM
Instructor-led, hands-on	**Kernel and driver development**	**Embedded Linux engineers**	**3 participants**

WHO IT IS FOR

- Kernel and driver developers moving from application or BSP work into kernel code.
- BSP and platform teams maintaining board support, Device Tree and drivers.
- Teams moving from vendor kernels to mainline and upstream contribution.

PREREQUISITES

- Linux command line and Git.
- C programming: pointers, structures and memory management.
- Embedded Linux basics: bootloader, kernel and root filesystem.
- A Linux host with a cross-compiler for the lab builds.

WHAT YOU WILL LEARN

- ✓ Navigate the kernel source tree and understand the release model.
- ✓ Configure, cross-compile and boot a kernel with U-Boot.
- ✓ Write, build, load and parameterise kernel modules.
- ✓ Describe hardware in the Device Tree and configure pin muxing.
- ✓ Follow the device and driver model through `probe()`.
- ✓ Write an I2C client driver and use character, regmap and devres frameworks.
- ✓ Work with memory, VFS, MMIO, clocks, scheduling, interrupts, locking and DMA.
- ✓ Debug the kernel and prepare a patch for upstream review.

On-site delivery only. The course is not available online, and sessions run for at least three participants. The programme can be tailored to your board and product.

PROGRAMME

The five-day programme

The suggested timing is five days. Each day combines lectures, kernel source walkthroughs and practical exercises; each block ends with a checkpoint and discussion.

DAY 1

Foundations

- Kernel introduction: role, system calls and pseudo filesystems
- Official, stable, LTS and vendor kernel sources
- Source tree, versioning, development model and release cycle
- Configuration, Kconfig and cross-compilation
- Booting with U-Boot, the command line and the kernel log

DAY 2

Modules and hardware description

- Kernel modules, utilities and autoloading
- Developing modules: hello module, symbols and parameters
- Kernel coding style and `checkpatch.pl`
- Device Tree syntax, bindings and validation
- Pin muxing and the pinctrl subsystem

DAY 3

Drivers

- Device and driver model: buses, platform devices and `probe()`
- Deferred probing and driver data structures
- I2C subsystem and an I2C client driver
- Character drivers, file operations and `ioctl`
- Input subsystem, regmap and devres

DAY 4

Kernel internals

- Memory management, allocators, reclaim and huge pages
- VFS, the page cache and file I/O paths
- Memory-mapped I/O and `ioremap()`
- Common Clock Framework and devfreq
- Processes, EEVDF scheduling and SMP

DAY 5

Concurrency, debugging and upstream

- Interrupts, threaded IRQs and deferred work
- Locking, RCU and atomics
- DMA principles, descriptors and the DMA APIs
- Debugging: `printk`, debugfs, oops decoding, ftrace, perf and BPF
- Upstream: preparing, posting and revising a patch

LEARN

- Short lectures and kernel source walkthroughs.
- Architecture and trade-offs explained in context.
- A checkpoint and discussion after each block.

BUILD

- A practical exercise after each block.
- Build and boot a kernel under QEMU or on a board.
- Write, load and debug modules and drivers.

Selected slides from the course deck are reproduced on the following pages. The complete deck is available online in the public course repository.

HANDS-ON LABS

Build, load and trace a driver

The labs follow the deck's examples: configure and build a kernel for a target, compile and load modules, describe hardware in the Device Tree and follow a driver through the device model.

```
$ make -C /path/to/kernel/sources M=$PWD
$ sudo insmod ./hello.ko
$ dmesg | tail -n 1
[  123.456789] Good morrow to this fair assembly.
$ sudo rmmod hello
$ dmesg | tail -n 1
[  123.456790] Alas, poor world, what treasure hast thou lost!
```

KERNEL SOURCES

- Configure and cross-compile the kernel for the target.
- Install it and boot it with U-Boot, then inspect the kernel log.

YOUR BOARD OR QEMU

- Run the exercises on a supported board or under QEMU.
- A Linux host with a cross-compiler and kernel headers is required.

WHAT THE EXERCISES COVER

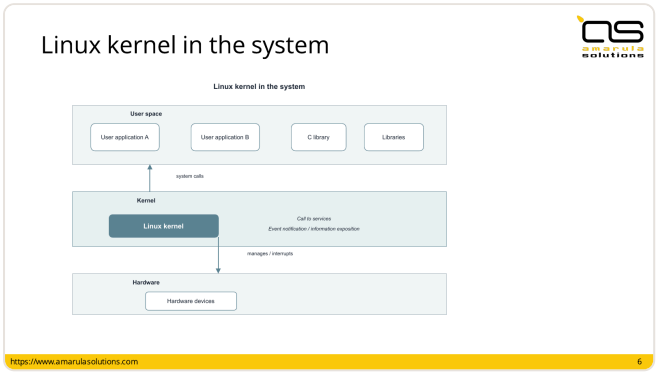
- ✓ Configure and cross-compile a kernel
- ✓ Write, build, load and unload a module
- ✓ Describe a device in the Device Tree
- ✓ Follow `probe()` through the device model
- ✓ Register a character device
- ✓ Debug with `printk`, debugfs and ftrace
- ✓ Install and boot the kernel with U-Boot
- ✓ Pass parameters to a module and read them from sysfs
- ✓ Configure pin muxing with pinctrl
- ✓ Write an I2C client driver
- ✓ Inspect memory, VFS and MMIO
- ✓ Prepare and post an upstream patch

> **Lab requirements:** a Linux host with a cross-compiler and kernel headers, QEMU or a supported board, and network access for the kernel sources.

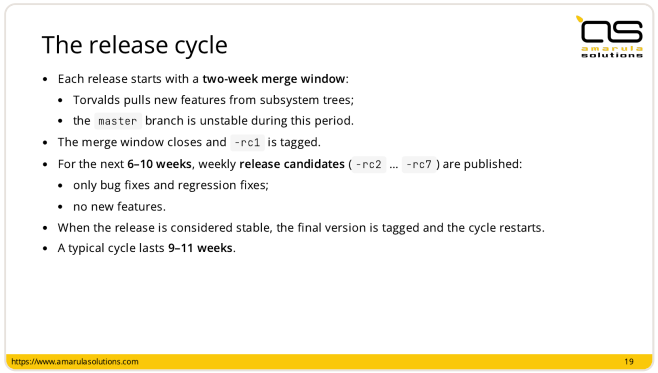
INSIDE THE COURSE

Selected slides

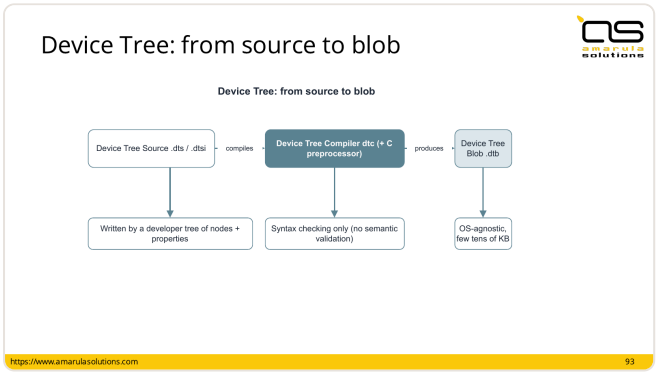
A few pages from the deck: the kernel in the system, the release cycle, Device Tree, character drivers, the interrupt path and upstream patch preparation.



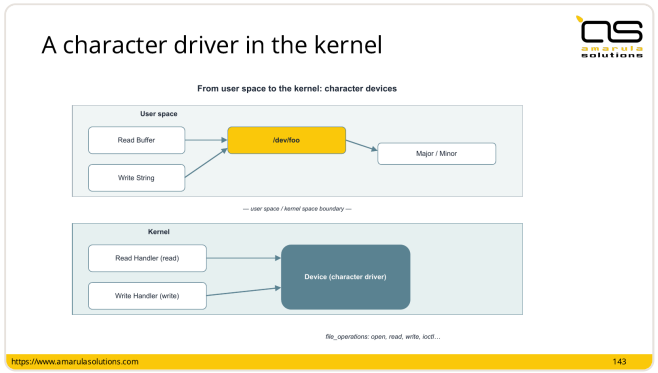
The kernel in the system. Hardware, kernel services, system calls and user space.



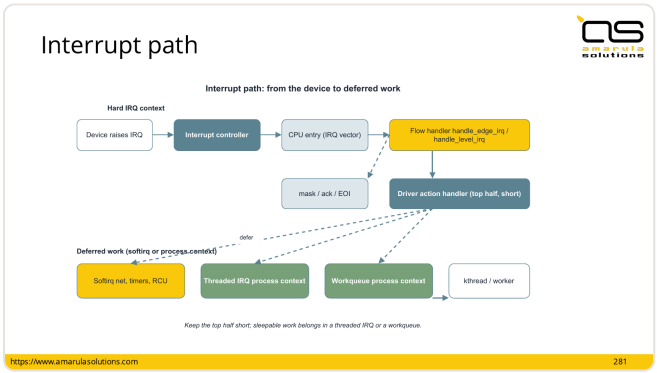
The release cycle. Merge windows, release candidates, stable and LTS branches.



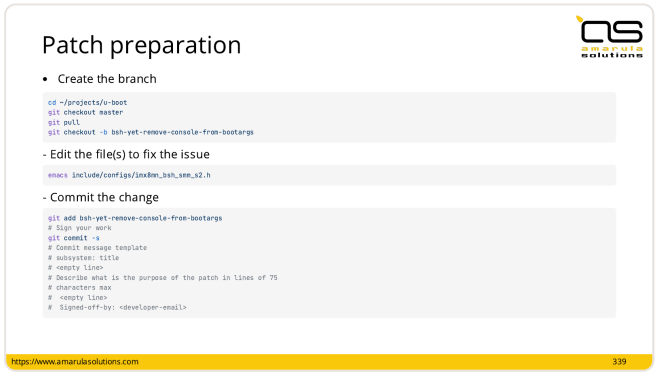
Device Tree. From `.dts` source through `dtc` to the `.dtb` consumed by the kernel.



Character drivers. From file operations in user space to the driver in the kernel.



Interrupts. From the device through the interrupt controller to deferred work.



Upstream. Preparing, signing off and posting a patch for review.

> The complete deck is available online in the **kernel-slides** repository at github.com/amarula/kernel-slides, licensed under GPL-2.0. The deck is inspired by Bootlin's Linux kernel and driver development training (CC BY-SA 3.0).

TRAINER AND BOOKING

Your trainer



Michael Trimarchi

Co-Founder & CEO, Amarula Solutions

Michael is a Linux software engineer working on embedded and real-time systems. He holds a Master's degree in Software Engineering from the University of Pisa and is co-author of the first version of the EDF patches submitted to mainline Linux.

Michael teaches this training from the perspective of a team that maintains mainline kernel support: how to read the source tree, choose the right subsystem framework and get a driver accepted upstream.

Book the course

DURATION	PARTICIPANTS	PRICE	TRAINER
5 days	**Minimum 3**	**On request**	**Michael Trimarchi**

On-site delivery only. Sessions are arranged on request; tell us about your board and preferred dates.

Request a quotation

Use the quotation form at amarulasolutions.com/quotation/ to tell us about your team and preferred dates.

Talk to us

Contact the Amarula team at amarulasolutions.com/contact/ or info@amarulasolutions.com.

© 2026 Amarula Solutions. Selected slides are excerpts from the course deck and remain the property of Amarula Solutions.