FOSDEM'26

Netboot without throwing a FIT

Kernel Devroom

Ahmad Fatoum – a.fatoum@pengutronix.de

About Me

Ahmad Fatoum

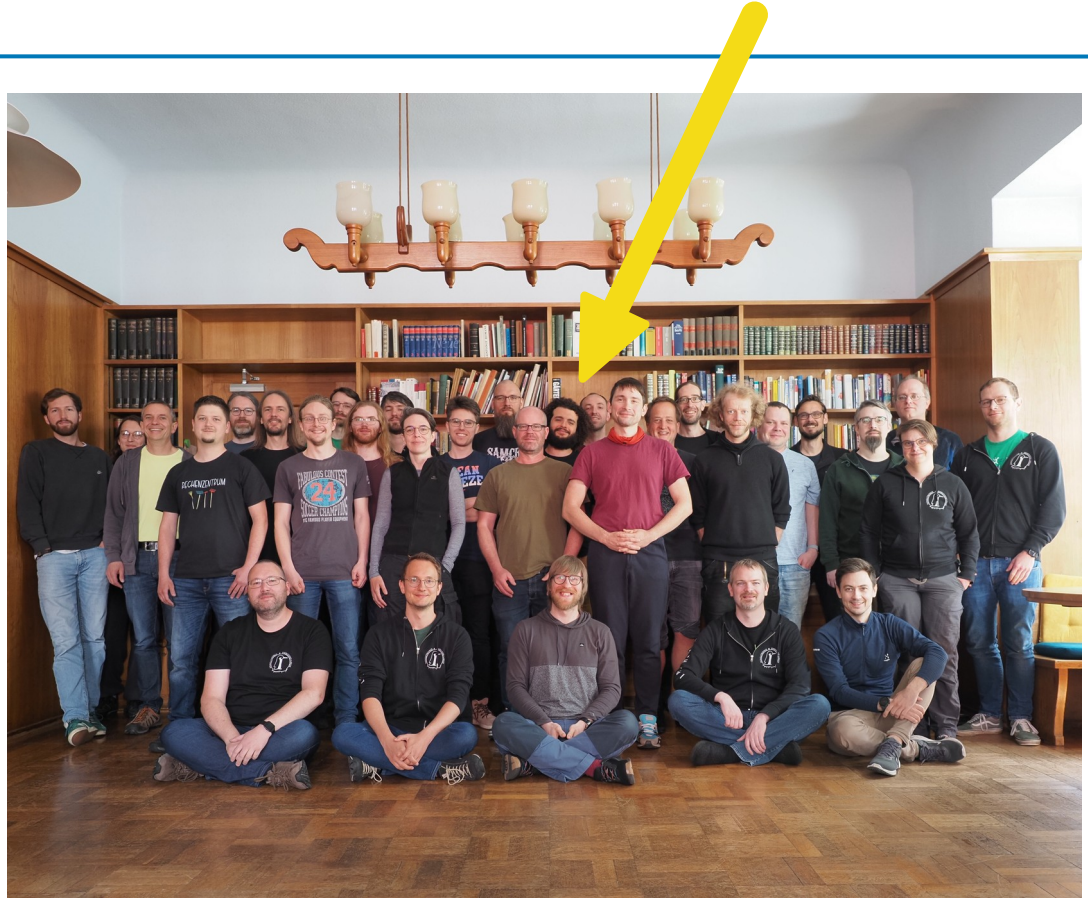
Pengutronix

a3f

a.fatoum@pengutronix.de

@a3f@fosstodon.org

- Kernel and Bootloader Porting
- Driver and Graphics Development
- System Integration
- Embedded Linux Consulting



My present netbooting

- Unpack the rootfs into an (NFS) exported directory
  - Self-describing: includes kernel, initramfs, DT and bootloader spec 🔗

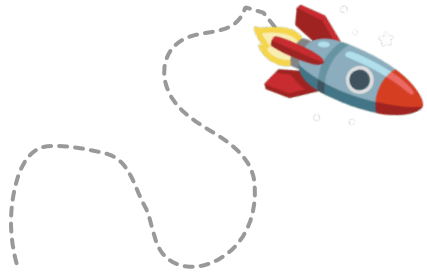
```
host$ cat /home/a3f/nfsroot/rock3a/loader/entries/rk3568-rock-3a.conf
title          PTXdist - Pengutronix-DistroKit rk3568-rock-3a
version        6.18
linux          /boot/Image
devicetree     /boot/rk3568-rock-3a.dtb
linux-appendroot  true
```

- Boot it! 🚀

```
barebox$ boot nfs://192.168.10.15:2049/home/a3f/nfsroot/rock3a
```

But some workarounds are needed

- User permissions are wrong when extracting without sudo
  - → Manually patch specific service
    e.g., `chmod -R gu-s`, service drop-ins
  - → Extract in fakeroot environment.
    Example: poky-nfsroot 🔗
  - → Patching NFS server to use map file?



- NFS may require network config changes (e.g. due to Ethernet Switches)
  - → Dynamically adapt configuration
  - → USB Ethernet Adapter
  - → usb9pfs since v6.12. Talk at FOSDEM 2025 🔗

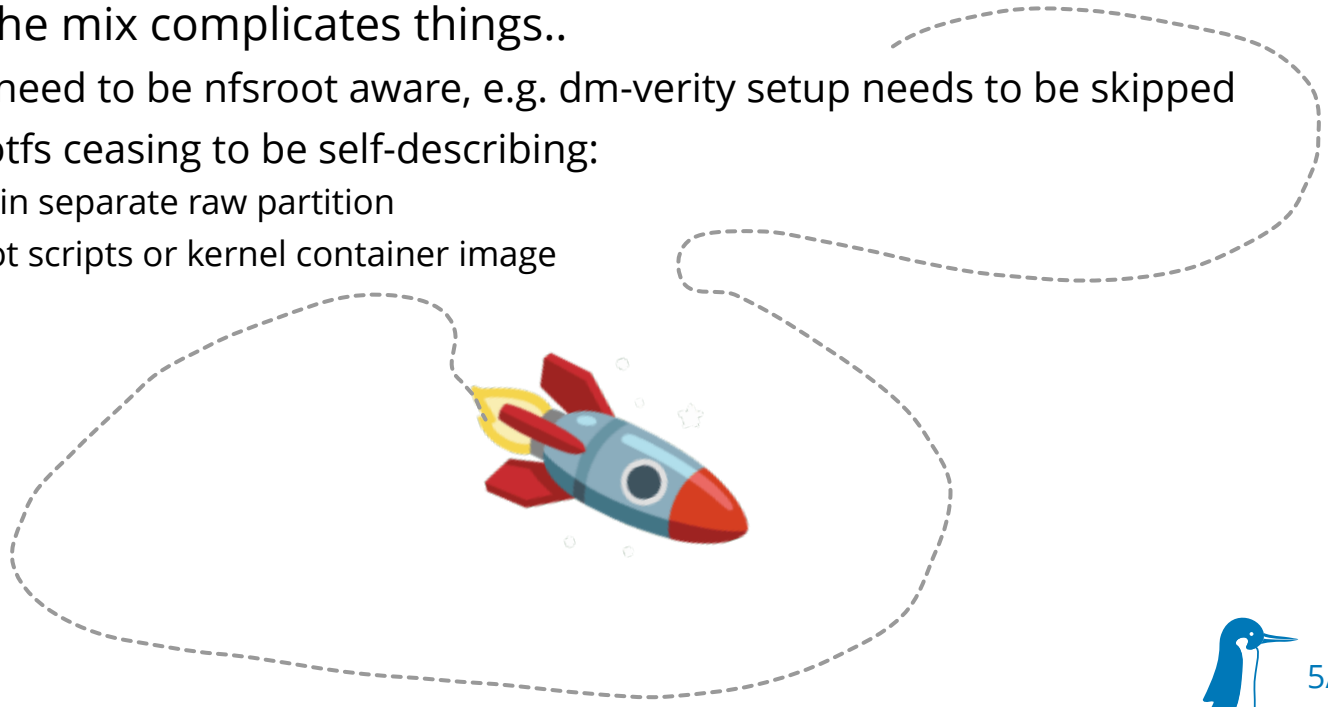
```
[Match]
Name=br0
KernelCommandLine=!nfsroot
```

- OS Build system has baked-in assumption about images (e.g., `ROOTFS_POSTPROCESS_COMMAND`)

Userspace says no

- Lots of hardcoded assumptions about the boot block device

  e.g., no concept of an active partition (important for A/B setups!)
- Adding verified boot to the mix complicates things..
  - Even more things that need to be nfsroot aware, e.g. dm-verity setup needs to be skipped
  - Often results in the rootfs ceasing to be self-describing:
    - Signed kernel image in separate raw partition
    - Logic moves into boot scripts or kernel container image



Userspace says no

- Lots of hardcoded assumptions about the boot block device

  e.g., no concept of an active partition (important for redundant setups!)
- Adding verified boot to the mix complicates things..
  - Even more things that need to be nfsroot aware, e.g. dm-verity setup needs to be skipped
  - Often results in the rootfs ceasing to be self-describing:
    - Signed kernel image in separate raw partition
    - Logic moves into boot scripts or kernel container image

Everyone needs to take care to keep it working, or..

Userspace says no

- Lots of hardcoded assumptions about the boot block device

  e.g., no concept of an active partition (important for redundant setups!)
- Adding verified boot to the mix complicates things..
  - Even more things that need to be nfsroot aware, e.g. dm-verity setup needs to be skipped
  - Often results in the rootfs ceasing to be self-describing:
    - Signed kernel image in separate raw partition
    - Logic moves into boot scripts or kernel partition

Everyone needs to take care to keep it working..

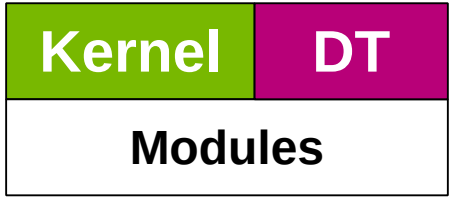
But I just want to network boot the kernel :(

- Reproducing kernel issues often needs full userspace
- What's the minimum we need to network boot only the kernel?

MVP: Minimum Viable (kernel netboot) Payload

1) Avoid messing with rootfs and OS build systems
   - Assumption: no out-of-tree modules

2) Network boot only the kernel and its inputs
   - Kernel
   - Device Tree
   - Modules



FIT: Flattened Image Tree

FIT

```
{
    description = "Linux-6.19.0-rc6";
    #address-cells = <0x1>;
    timestamp = <0x697a0898>;
    configurations {
        conf-11 {
             compatible = "fsl,imx8mm-ddr4-evk",
                          "fsl,imx8mm";
             description = "FSL i.MX8MM DDR4 EVK";
             kernel = "kernel";
             ramdisk = "ramdisk";
             fdt = "fdt-11";
         };
         conf-419 {
             compatible = "radxa,rock3a",
                          "rockchip,rk3568";
             description = "Radxa ROCK 3A";
             kernel = "kernel";
             ramdisk ="ramdisk";
             fdt = "fdt-419";
        };
    };
};
```

FIT

```
images {
     kernel {
          /* … */
     };
     ramdisk {
          /* … */
     };
     fdt-11 {
          /* … */
     };
     fdt-419 {
          /* … */
     };
};
```

https://fitspec.osfw.foundation

FIT: Matching

FIT

```
{
    description = "Linux-6.19.0-rc6";
    #address-cells = <0x1>;
    timestamp = <0x697a0898>;
    configurations {
        conf-11 {
             compatible = "fsl,imx8mm-ddr4-evk",
                          "fsl,imx8mm";
             description = "FSL i.MX8MM DDR4 EVK";
             kernel = "kernel";
             ramdisk = "ramdisk";
             fdt = "fdt-11";
         };
         conf-419 {
             compatible = "radxa,rock3a",
                          "rockchip,rk3568";
             description = "Radxa ROCK 3A";
             kernel = "kernel";
             ramdisk ="ramdisk";
             fdt = "fdt-419";
        };
    };
};
```

Bootloader DT

```
/ {
    compatible = "radxa,rock3a",
                 "rockchip,rk3568"

    /* 6700~ more lines omitted */
};
```

❌ **No match**

FIT: Matching

FIT

```
{
    description = "Linux-6.19.0-rc6";
    #address-cells = <0x1>;
    timestamp = <0x697a0898>;
    configurations {
        conf-11 {
            compatible = "fsl,imx8mm-ddr4-evk",
                         "fsl,imx8mm";
            description = "FSL i.MX8MM DDR4 EVK";
            kernel = "kernel";
            ramdisk = "ramdisk";
            fdt = "fdt-11";
        };
        conf-419 {
            compatible = "radxa,rock3a",
                         "rockchip,rk3568";
            description = "Radxa ROCK 3A";
            kernel = "kernel";
            ramdisk ="ramdisk";
            fdt = "fdt-419";
        };
    };
};
```

Bootloader DT

```
/ {
    compatible = "radxa,rock3a",
                 "rockchip,rk3568"

    /* 6700~ more lines omitted */
};
```

✅ **Match!**

FIT: Booting

FIT

```
{
    description = "Linux-6.19.0-rc6";
    #address-cells = <0x1>;
    timestamp = <0x697a0898>;
    configurations {
        conf-11 {
            compatible = "fsl,imx8mm-ddr4-evk",
                         "fsl,imx8mm";
            description = "FSL i.MX8MM DDR4 EVK";
            kernel = "kernel";
            ramdisk = "ramdisk";
            fdt = "fdt-11";
        };
        conf-419 {
            compatible = "radxa,rock3a",
                         "rockchip,rk3568";
            description = "Radxa ROCK 3A";
            kernel = "kernel";
            ramdisk ="ramdisk";
            fdt = "fdt-419";
        };
    };
};
```

FIT

```
images {
    kernel {
        description = "Linux-6.19.0-rc6";
        type = "kernel_noload";
        arch = "arm64";
        os = "linux";
        compression = "gzip";
        data = /* 10281261 bytes omitted */;
        load = <0x0>;
        entry = <0x0>;
    };
    ramdisk {
        description = "Ramdisk";
        type = "ramdisk";
        arch = "arm64";
        compression = "none";
        os = "linux";
        data = /* 2020830 bytes omitted */;
    };
    fdt-11 {
        description = "imx8mm-ddr4-evk.dtb";
        type = "flat_dt";
        arch = "arm64";
        compression = "gzip";
        data = /* 9777 more bytes omitted */;
    };
    fdt-419 {
        description = "rk3568-rock-3a.dtb";
        type = "flat_dt";
        arch = "arm64";
        compression = "gzip";
        data = /* 13423 more bytes omitted */;
    };
};
```

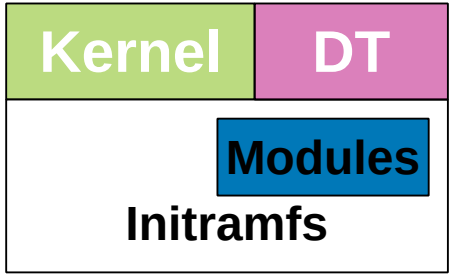
FIT in Linux

- Since v6.10: `make ARCH=arm64 image.fit`
  - Contains kernel and all enabled device trees

What about modules?

- Possible mismatch with rootfs
- But we can’t build everything into the kernel
- Solution: Linux transparently handles concatenated CPIOs
- Let’s install modules into an initramfs!
  - Since v6.19: make modules-cpio-pkg
  - Soon hopefully: initrd inclusion into FIT by Simon Glass:

    `[PATCH v9 0/6] scripts/make_fit: Support ramdisks and faster operations` 🔗



- But who loads to the modules?

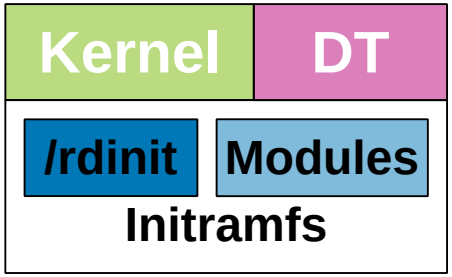
Take care of some quirks with `modules-cpio-pkg`:

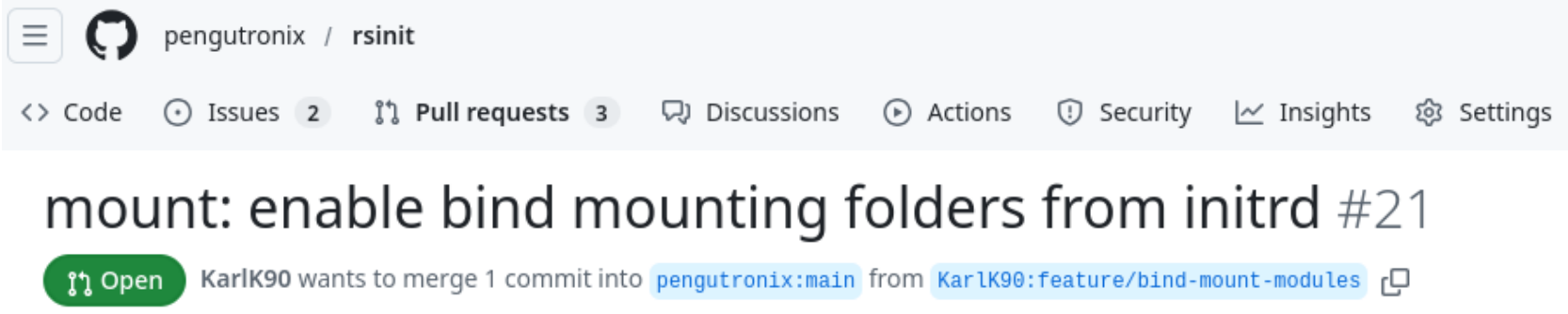
- Modules are not stripped → Enable `CONFIG_DEBUG_INFO_SPLIT`
- Modules are `-rw------` → `chmod go+r modules-*`

Making use of initrd modules

- Get the initramfs modules into the rootfs

```
$ mount -o bind /lib/modules ${NEW_ROOT_MOUNT}/lib/modules
```





by Stefan Kerkmann

Putting it all together

```
make all modules-cpio-pkg

modules="$(echo modules-"$(make kernelrelease)"-*.cpio)"

gzip -f $cpio
cat ${cpio}.gz rsinit.cpio.gz >modules.cpio.gz
```

```
make image.fit FIT_EXTRA_ARGS=--ramdisk=modules.cpio.gz
```

- But: Bootloader does more than mere booting, e.g.:
  - Command line fixups (`root=`, `console=`, … etc.)
  - Apply fixups and overlays to Device Tree

→ Bootloader integration would be *nice*

A last missing puzzle piece

- barebox supports late override of boot artifacts:

  Boot as usual, but at the very end switch out boot artifacts

  e.g., replace the Device Tree or the initramfs

- Let's put it on steroids
  - Replace individual artifacts from a FIT image
  - Support replacing the kernel image
  - Support appending the initrd on the fly
  - Fetch the boot override description over the network

  Upstreaming in progress. Current state 🔗

https://barebox.org 🔗

It's demo time

Thanks for listening!

The FIT image generated on Slide 16

Future Outlook:

- Network Block Device as alternative to NFS?
- With barebox newly supporting dm-verity, self-describing rootfs with bootloader specification may be in reach again

Questions?