Yocto Project Summit 2021

Designing OSTree based embedded Linux systems with the Yocto Project

Sergio Prado

Toradex

$ WHOAMI

- Designing and developing embedded software for 25+ years.
- Software Team Lead at Toradex (https://www.toradex.com/).
- Consultant/trainer at Embedded Labworks (e-labworks.com/en).
- Open source software contributor, including Buildroot, Yocto Project and the Linux kernel.
- Sometimes write technical stuff at https://embeddedbits.org/.
- Social networks:
  Twitter: @sergioprado
  Linkedin: https://linkedin.com/in/sprado

AGENDA

1. Introduction to OSTree
2. Booting and running an OSTree-based system
3. Building an OSTree-based system with meta-updater
4. Remote updates with OSTree-based systems

WHAT IS OSTREE?

- OStree, also known as **libostree**, provides a "git-like" model for committing and downloading bootable filesystem trees (rootfs).
- It’s like Git, in a sense that it stores checksum'ed files (SHA256) in a content-addressed object-store.
- It’s different from Git, because files are checked out via hard links, and they are immutable (read-only) to prevent corruption.
- Designed and currently maintained by Colin Walters (GNOME, OpenShift, RedHat CoreOS developer)

A FEW OSTREE USERS

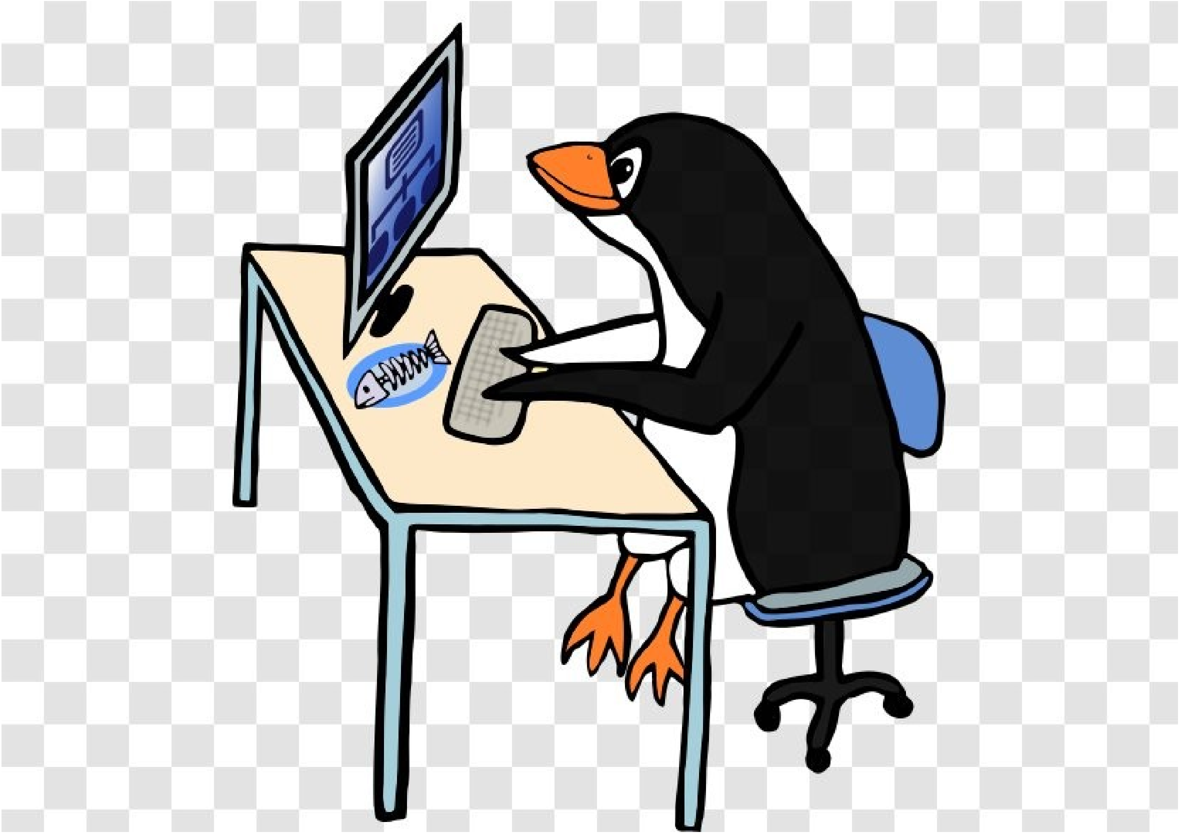
- Linux distributions:
  - GNOME Continuous, Gnome OS
  - Fedora CoreOS, Fedora Silverblue, Fedora IoT
  - Endless OS
  - Linux microPlatform
  - TorizonCore
- Package management systems:
  - rpm-ostree
  - flatpak

OSTREE IN A NUTSHELL

- A Git-like content-addressed object store, where we can store individual files or full filesystem trees.
- Provides a mechanism to commit and checkout branches (or "refs").
- Manages bootloader configuration via The Boot Loader Specification, a standard on how different operating systems can cooperatively manage a boot loader configuration (GRUB and U-Boot supported).

  https://www.freedesktop.org/wiki/Specifications/BootLoaderSpec/
- It operates entirely in userspace via a library and CLI tools, and will work on top of any Linux filesystem.

HANDS-ON 1: OSTREE



USING OSTREE AS A ROOTFS (1)

- In the main storage partition, we have basically two directories, the boot directory (`/boot`) and the OSTree repository (`/ostree`), mounted at `/sysroot`.

- Filesystem trees (also called deployments) are checked out at `/sysroot/ostree/deploy/<os>/deploy/<commit>/` (files there are just hard links to objects in the repository).

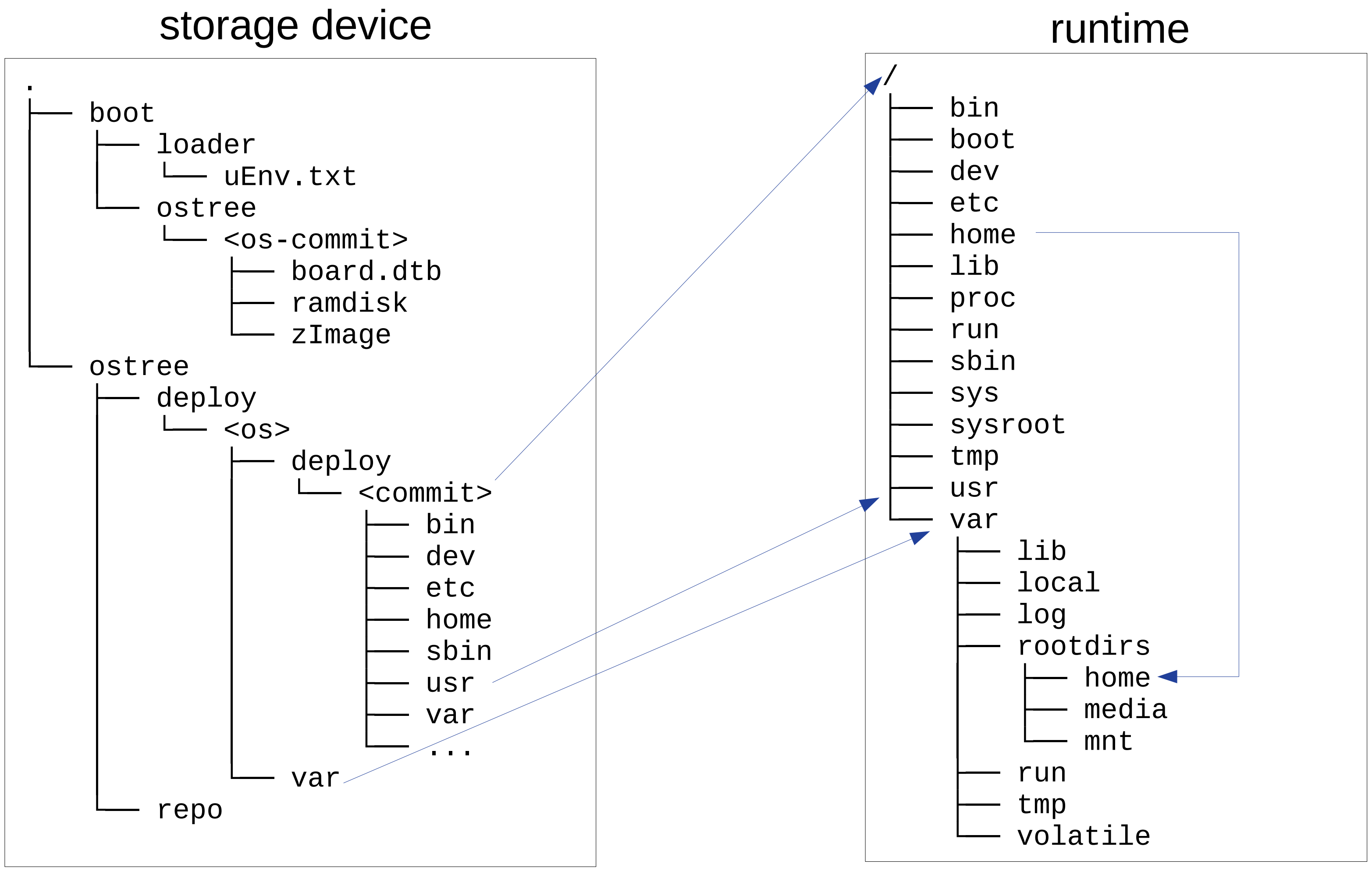
- A deployment is bind-mounted as a read-write rootfs at `/`, and the `/usr` directory from the deployment is bind-mounted read-only at `/usr`.

USING OSTREE AS A ROOTFS (2)

- Runtime generated data should go to `/var` (bind mounted at `/sysroot/ostree/deploy/<os>/var/`) and other writable/persistent directories also links to `/var` (e.g. `/home` -> `/var/rootdirs/home`).

- Operating system configuration (`/etc`) is handled in a special way (it starts with the content of `/usr/etc`, but you can write to it, and the changes are kept during new deployments).

OSTREE FILESYSTEM LAYOUT (SIMPLIFIED)



storage device

```
.
├── boot
│   ├── loader
│   │   └── uEnv.txt
│   └── ostree
│       └── <os-commit>
│           ├── board.dtb
│           ├── ramdisk
│           └── zImage
└── ostree
    ├── deploy
    │   └── <os>
    │       ├── deploy
    │       │   └── <commit>
    │       │       ├── bin
    │       │       ├── dev
    │       │       ├── etc
    │       │       ├── home
    │       │       ├── sbin
    │       │       ├── usr
    │       │       ├── var
    │       │       └── ...
    │       └── var
    └── repo
```

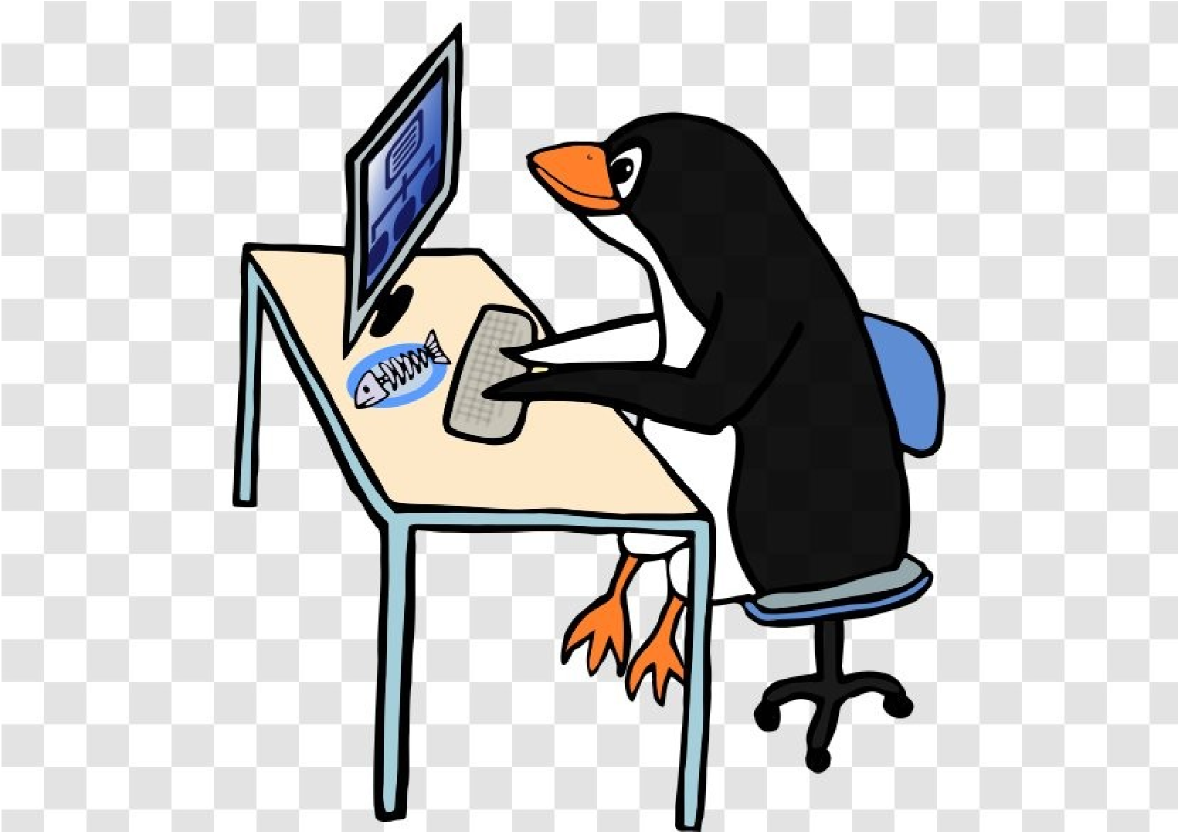
runtime

```
/
├── bin
├── boot
├── dev
├── etc
├── home
├── lib
├── proc
├── run
├── sbin
├── sys
├── sysroot
├── tmp
├── usr
└── var
    ├── lib
    ├── local
    ├── log
    ├── rootdirs
    │   ├── home
    │   ├── media
    │   └── mnt
    ├── run
    ├── tmp
    └── volatile
```

DEPLOYING A NEW OS

- A new deployment directory from a OSTree commit is created at `/sysroot/ostree/deploy/<os>/deploy/<commit>/`.
- OSTree performs a 3-way merge in `/etc` using 1) the old default configuration, 2) the current configuration and 3) the new default configuration.
- Kernel artifacts (kernel, device tree, ramdisk, etc) are copied from the deployment to `/boot/ostree/<os>-<commit>`, and bootloader configuration files may be changed.

HANDS-ON 2: BOOTING/RUNNING WITH OSTREE



OSTREE INTEGRATION

1. Generate the sysroot partition with the boot directory (`/boot`) and the OSTree repository (`/ostree`).

2. Prepare the default deployment in `/sysroot/ostree/deploy/<os>/deploy/<commit>/`.

3. Make sure U-Boot will be able to load and boot the kernel artifacts (kernel image, device tree, ramdisk).

4. Boot a ramdisk image that will mount the OSTree deployment and switch to it.

5. Make sure to follow OSTree "requirements": UsrMove, imuttable system (`/usr` is read-only), OS configuration in `/etc`, data in `/var`.

The first 4 steps are already (mostly) implemented in `meta-updater`!

META-UPDATER

- Yocto Project/OpenEmbedded layer for OSTree-based systems.
- Includes a client for remote updates called Aktualizr, based on the Uptane standard.
- Configurable via variables that can be defined in a configuration file.

  https://docs.ota.here.com/ota-client/latest/build-configuration.html
- Supported platforms include QEMU, Raspberry Pi, Intel Minnowboard, BeagleBone Black, etc; and adding support to new platforms is not hard.

  https://docs.ota.here.com/ota-client/latest/bsp-integration.html

META-UPDATER INTEGRATION

- Create a board class for the machine (`sota_{MACHINE}.bbclass`), defining kernel image type to be used, kernel command line parameters, boot script name, etc.

  https://docs.ota.here.com/ota-client/latest/add-board-class.html

- Generate a physical image with the partitions in the correct place for OSTree compatibility (the most common approach is to use Wic for that).

  https://docs.ota.here.com/ota-client/latest/setup-boot-image-for-ostree.html

- Adapt distro to OSTree, like installing everything inside `/usr` (`DISTRO_FEATURE += "usrmerge"`), enable the needed filesystem types (`ota-ext4 ostree.tar.bz2 ota.tar.xz wic`), create boot script for inicialization.

  https://docs.ota.here.com/ota-client/latest/add-meta-updater-to-vendors-sdk.html

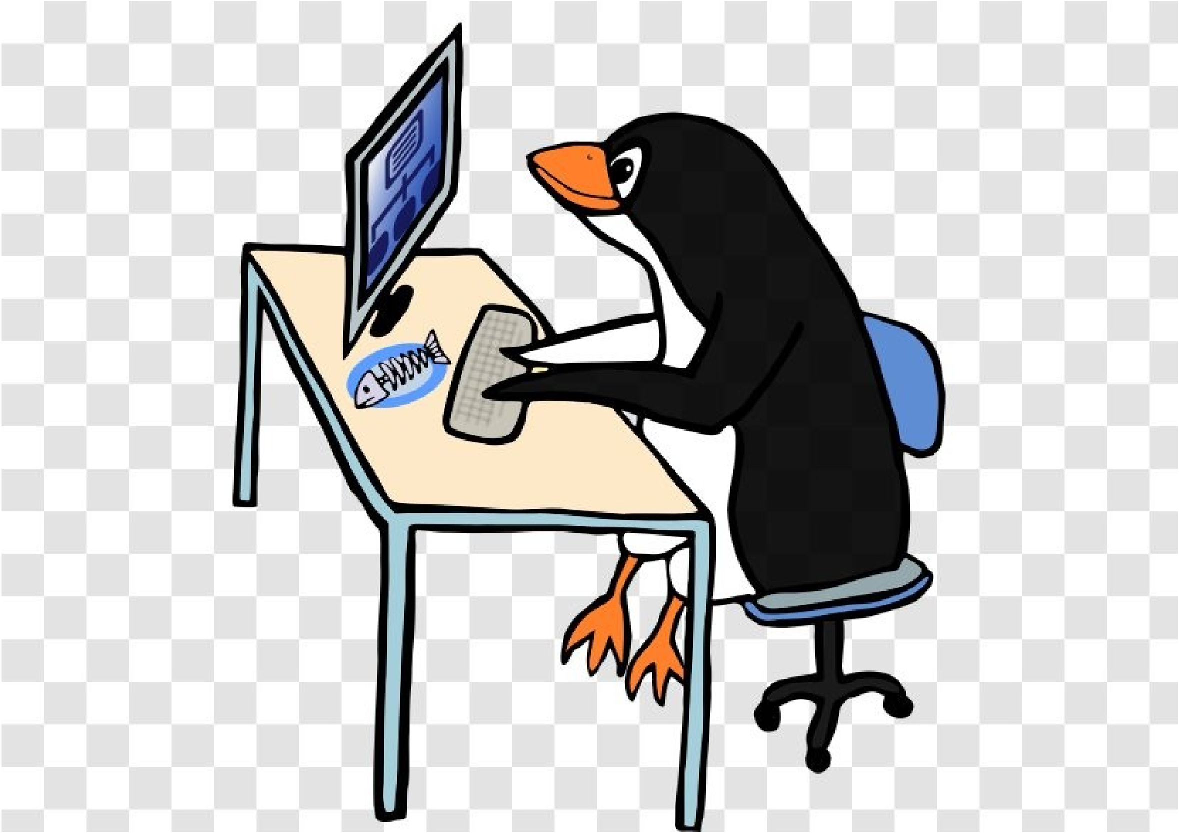
REMOTE UPDATE SYSTEMS

- **Package-based**: Low bandwidth but unreliable and difficult to manage.
- **Partition-based**: Robust but consumes a lot of network bandwidth and storage.
- **Atomic differential**: Combines robustness with minimal bandwidth and storage consumption, adding some complexity to the operating system.

OSTREE IN AN UPDATE SYSTEM

- Atomic
- Delta-based
- On-the-fly
- Updates via HTTP
- Commits and deltas can be signed

HANDS-ON 3: UPDATING WITH OSTREE



OSTREE TRADE-OFFS

- OSTree is a very nice technology, but...
- OSTree adds complexity to the system, and we need to comply to its requirements.
- Since there is only one physical filesystem, the system may become unbootable if it gets corrupted due to hardware bugs, driver bugs, etc.
- Rollback logic is not part of OSTree, and should be implemented separately, ideally in the bootloader.

LINKS

- OSTree project's repository:
  https://github.com/ostreedev/ostree

- OSTree documentation:
  https://ostreedev.github.io/ostree/

- meta-updater layer:
  https://github.com/advancedtelematic/meta-updater

Q&A

Sergio Prado
sergio@embeddedbits.org

https://twitter.com/sergioprado
https://www.linkedin.com/in/sprado

Thank you!