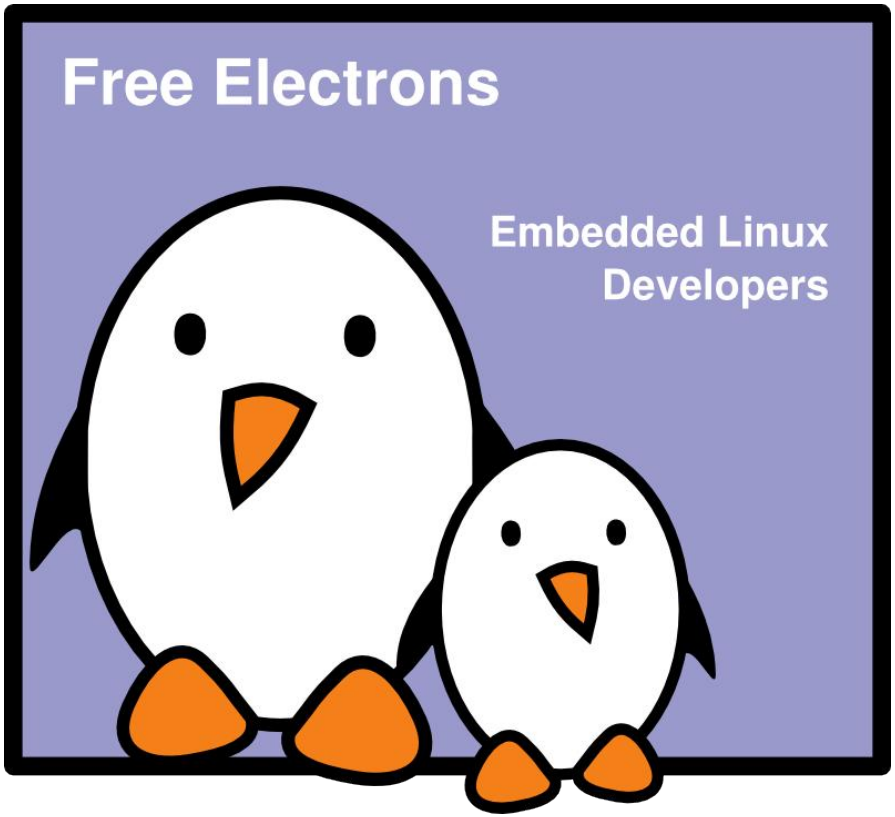
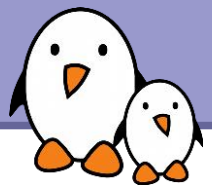
Flash filesystems

Michael Opdenacker
Thomas Petazzoni
Free Electrons



Latest update: Dec 20, 2010,
Document sources, updates and translations:
http://free-electrons.com/docs/flash-filesystems
Corrections, suggestions, contributions and translations are welcome!



The MTD subsystem

MTD: Memory Technology Devices (flash, ROM, RAM)

Linux filesystem interface

MTD “User” modules

UBI | jffs2 | Char device | Block device

yaffs2 | Read-only block device

Flash Translation Layers
for block device emulation
Caution: patented algorithms!

FTL | NFTL | INFTL

MTD Chip drivers

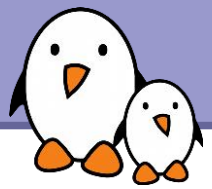
NOR flash | RAM chips

NAND flash | DiskOnChip flash | ROM chips

Block device | Virtual memory

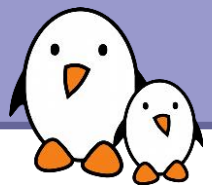
Virtual devices appearing
as MTD devices

Memory devices
hardware



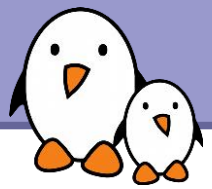
MTD devices

- MTD devices are visible in `/proc/mtd`
- The **mtdchar** driver creates a character device for each MTD device of the system
  - Usually named `/dev/mtdX`, major 90. Even minors for read-write access, odd minors for read-only access
  - Provide `ioctl()` to erase and manage the flash
  - Used by the *mtd-utils*
- The **mtdblock** driver creates a block device for each MTD device of the system
  - Usually named `/dev/mtdblockX`, major 31. Minor is the number of the MTD device
  - Allows read/write block-level access. But bad blocks are not handled, and no wear leveling is done for writes.



MTD partitioning

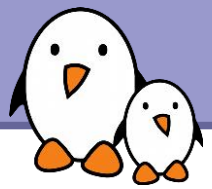
- MTD devices are usually partitioned
  - It allows to use different areas of the flash for different purposes : read-only filesystem, read-write filesystem, backup areas, bootloader area, kernel area, etc.
- Unlike block devices, which contains their own partition table, the partitioning of MTD devices is described externally
  - Hard-coded into the kernel code
  - Specified through the kernel command line
- Each partition becomes a separate MTD device
  - Different from block device labeling (hda3, sda2)
  - `/dev/mtd1` is either the second partition of the first flash device, or the first partition of the second flash device



Definition of MTD partitions

MTD partitions are defined in the kernel, in the board definitions:
`arch/arm/mach-at91/board-usb-a9263.c` example:

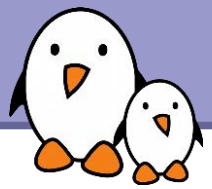
```
static struct mtd_partition __initdata ek_nand_partition[] = {
        {
                .name   = "Linux Kernel",
                .offset = 0,
                .size   = SZ_16M,
        },
        {
                .name   = "Root FS",
                .offset = MTDPART_OFS_NXTBLK,
                .size   = 120 * SZ_1M,
        },
        {
                .name   = "FS",
                .offset = MTDPART_OFS_NXTBLK,
                .size   = 120 * SZ_1M,
        }
};
```



Modifying MTD partitions (1)

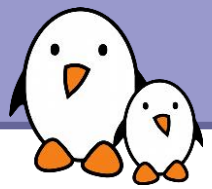
- MTD partitions can fortunately be defined through the kernel command line.
- First need to find the name of the MTD device. Look at the kernel log at boot time:

```
NAND device: Manufacturer ID: 0xec, Chip ID:
0xda (Samsung NAND 256MiB 3,3V 8-bit)
Scanning device for bad blocks
Bad eraseblock 2000 at 0x0fa00000
Creating 3 MTD partitions on "atmel_nand":
0x00000000-0x01000000 : "Linux Kernel"
0x01000000-0x08800000 : "Root FS"
0x08800000-0x10000000 : "FS"
```



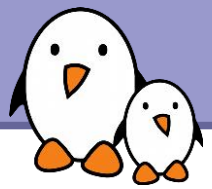
Modifying MTD partitions (2)

- You can now use the `mtdparts` kernel boot parameter
- Example:
  `mtdparts=atmel_nand:2m(kernel)ro,1m(rootfs)ro,-(data)`
- We've just defined 3 partitions in the `atmel_nand` device:
  - `kernel` (2M)
  - `rootfs` (1M)
  - `data`
- Partition sizes must be multiple of the erase block size.
  You can use sizes in hexadecimal too. Remember the below sizes:
  `0x20000` = 128k, `0x100000` = 1m, `0x1000000` = 16m
- `ro` lists the partition as read only
- `-` is used to use all the remaining space.



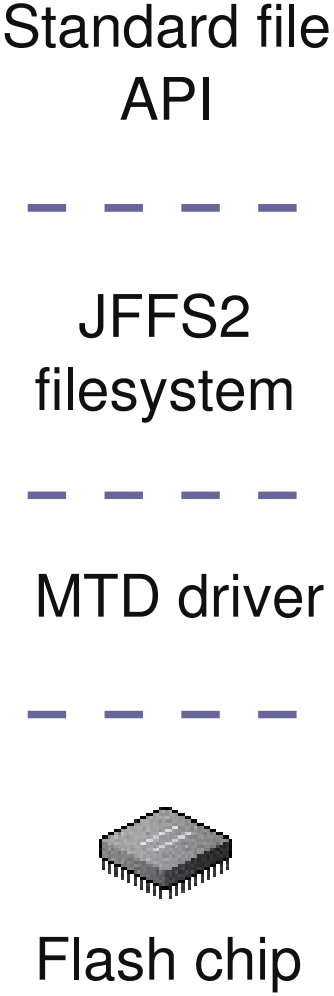
mtd-utils

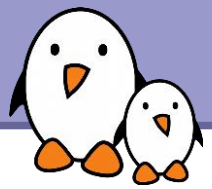
- A set of utilities to manipulate MTD devices
  - `mtdinfo` to get detailed information about a MTD device
  - `flash_eraseall` to completely erase a given MTD device
  - `flashcp` to write to NOR flash
  - `nandwrite` to write to NAND flash
  - UBI utilities
  - Flash filesystem image creation tools: `mkfs.jffs2`, `mkfs.ubifs`
- Usually available as the mtd-utils package in your distribution
- See http://www.linux-mtd.infradead.org/



jffs2

- Supports both NAND and NOR flash
- Today's standard filesystem for MTD flash
- Nice features: on the fly compression (saves storage space and reduces I/O), power down reliable, wear-leveling and ECC.
- Drawbacks: doesn't scale well
  - Mount time depending on filesystem size: the kernel has to scan the whole filesystem at mount time, to read which block belongs to each file.
  - Need to use the `CONFIG_JFFS2_SUMMARY` kernel option to store such information in flash. This dramatically reduces mount time (from 16 s to 0.8s for a 128 MB partition).

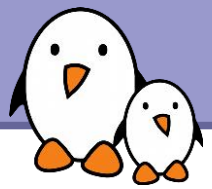




jffs2 - How to use

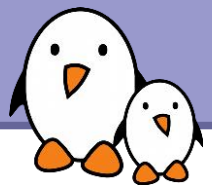
On the Linux **target**

- Need either the `mtd-utils` from the MTD project, or their embedded variants from Busybox
- Erase and format a partition with jffs2:
  `flash_eraseall -j /dev/mtd2`
  Mount the partition:
  `mount -t jffs2 /dev/mtdblock2 /mnt/flash`
  Fill the contents by writing
  (copying from NFS or from external storage)
- Other possibility: use a jffs2 image (see next page to produce it):
  `flash_eraseall /dev/mtd2`
  `nandwrite -p /dev/mtd2 rootfs.jffs2`



How to create a jffs2 image

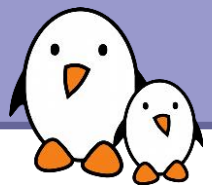
- `mkfs.jffs2` command available in the `mtd-utils` package.
  Caution: unlike some `mkfs` commands, it doesn't create a filesystem, but a filesystem image.
- First, find the erase block size from U-boot `nand info`:
  ```
  Device 0: NAND 256MiB 3,3V 8-bit, sector size 128 KiB
  ```
- Then create the image on your workstation:
  ```
  mkfs.jffs2 --pad --no-cleanmarkers
  --eraseblock=128 -d rootfs/ -o rootfs.jffs2
  ```
- The `--pad` option pads the jffs2 image contents until the end of the final erase block.
- It is fine if the jffs2 image is smaller than the MTD partition. The jffs2 file system will use the entire partition anyway.
- The `--no-cleanmarkers` option is for NAND flash only.



Mounting a jffs2 image on your host

Useful to edit `jffs2` images on your development system
Mounting an MTD device as a loop device is a bit complex task.
Here's an example for `jffs2`, for your reference:

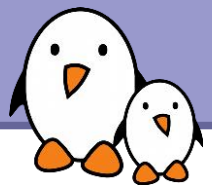
- First find the erase block size used to create the jffs2 image.
  Let's assume it is 128KiB (131072 bytes).
- Create a block device from the image
  `losetup /dev/loop0 root.jffs2`
- Emulate an MTD device from a block device,
  using the block2mtd kernel module
  `modprobe block2mtd block2mtd=/dev/loop0,131072`
- Finally, mount the filesystem (create `/mnt/jffs2` if needed)
  `mount -t jffs2 /dev/mtdblock0 /mnt/jffs2`



Initializing jffs2 partitions from U-boot

You may not want to have `mtd-utils` on your target!

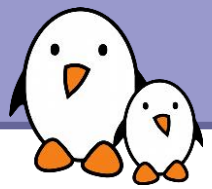
- Create a JFFS2 image on your workstation
- In the U-Boot prompt:
  - Download the jffs2 image to RAM with `tftp`
    Or copy this image to RAM from external storage
    (U-boot understands FAT filesystems and supports USB storage)
  - Flash it inside an MTD partition
    (exact instructions depending on flash type, NOR or NAND,
    reuse the instructions used to flash your kernel). Make sure to write only the size of the image, not more!
  - If you boot on a jffs2 root filesystem, add `root=/dev/mtdblock<x>` and `rootfstype=jffs2` to the Linux command line arguments.
- Limitation: need to split the jffs2 image in several chunks
  if bigger than the RAM size.



yaffs2

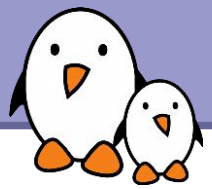
http://www.yaffs.net/

- Mainly supports NAND flash
- No compression
- Wear leveling, ECC, power failure resistant
- Fast boot time
- Code available separately through `git`
  (Dual GPL / Proprietary license
  for non Linux operating systems)



yaffs2 - How to use

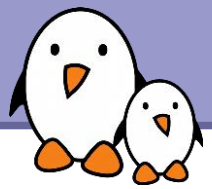
- Erase a partition:
  `flash_eraseall /dev/mtd2`
- The filesystem is automatically formatted at the first mount:
  `mount -t yaffs2 /dev/mtdblock2 /mnt/flash`
- Images can be created with mkyaffs tool, from yaffs-utils
  http://www.aleph1.co.uk/cgi-bin/viewvc.cgi/yaffs/utils/



UBI (1)

Unsorted Block Images

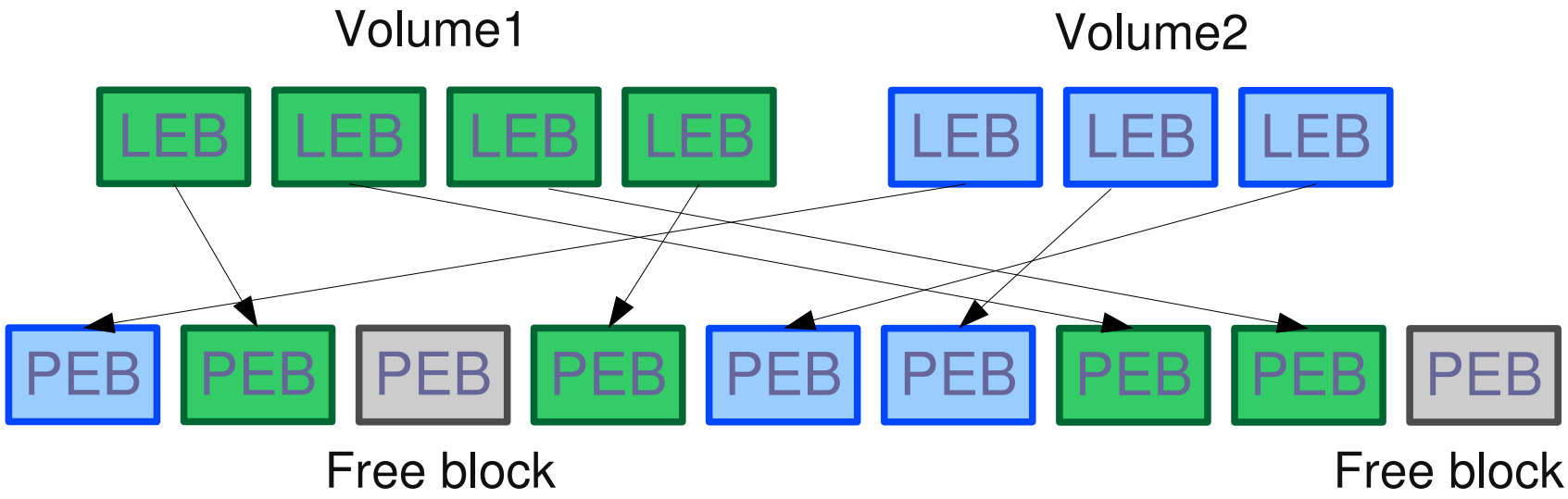
- http://www.linux-mtd.infradead.org/doc/ubi.html
- Volume management system on top of MTD devices.
- Allows to create multiple logical volumes and spread writes across all physical blocks.
- Takes care of managing the erase blocks and wear leveling. Makes filesystem easier to implement.

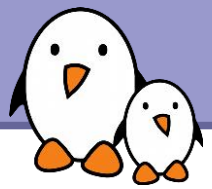


UBI (2)

UBI
Logical
Erase Blocks

MTD
Physical
Erase Blocks

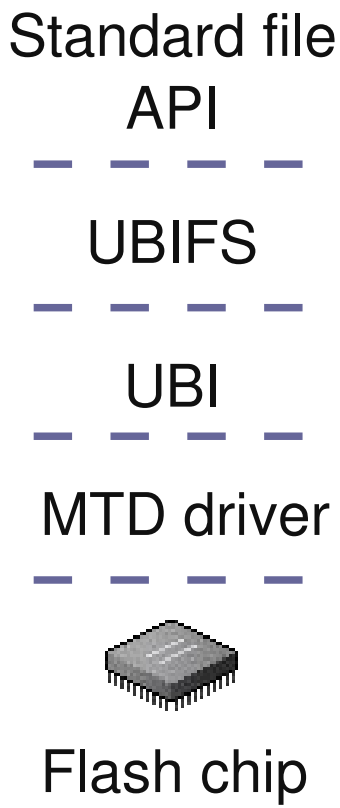


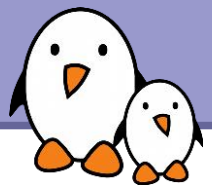


UBIFS

http://www.linux-mtd.infradead.org/doc/ubifs.html

- The next generation of the jffs2 filesystem, from the same linux-mtd developers.
- Available in Linux 2.6.27
- Works on top of UBI volumes
- Has a noticeable metadata overhead on very small partitions (4M, 8M)



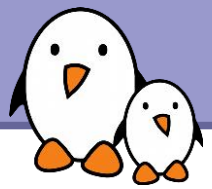


UBI - Preparation

- Erase your flash partition while preserving your erase counters
  `ubiformat /dev/mtd1`
  See http://www.linux-mtd.infradead.org/faq/ubi.html if you face problems
- Need to create a `/dev/ubi_ctrl` char device (if you don't have udev)
  - This special character device is used by other UBI utilities
  - Major and minor number allocated in the kernel. Find these numbers in `/sys/class/misc/ubi_ctrl/dev` (e.g.: `10:63`)
  - Or run `ubinfo`:

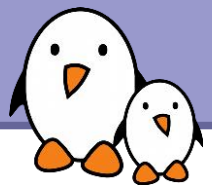
```
UBI version:                    1
Count of UBI devices:           1
UBI control device major/minor: 10:63
Present UBI devices:            ubi0
```

- These steps are done once for all



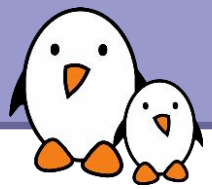
UBI - Attaching

- Attach UBI to one (of several) of the MTD partitions:
  `ubiattach /dev/ubi_ctrl -m 1`
- This command creates the ubi0 device, which represent the full UBI space stored on MTD device 1
  - Find the major and minor numbers used by UBI:
    `cat /sys/class/ubi/ubi0/dev` (e.g. `253:0`)
  - Create the UBI device file:
    `mknod /dev/ubi0 c 253 0`
  - This UBI space can contain several volumes



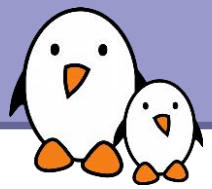
UBI - Volume management

- Volume creation with `ubimkvol`
  - `ubimkvol /dev/ubi0 -N test -s 116MiB`
  - `ubimkvol /dev/ubi0 -N test -m` (max available size)
  - The volume is then identified as `ubi0:test` for the mount/umount commands
- Volume removal with `ubirmvol`
  - `ubirmvol /dev/ubi0 -N test`



UBIFS - How to use

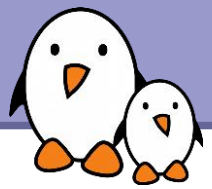
- When a UBI volume is created, creating an empty UBIFS filesystem is just a matter of mounting it
  - `mount -t ubifs ubi0:test /mnt/flash`
- Images of UBIFS filesystems can be created using the `mkfs.ubifs` utility
  - `mkfs.ubifs -m 512 -e 128KiB -c 100 -r /opt/img ubifs.img`
  - Can be written to a UBI volume using `ubiupdatevol` and the `/dev/ubiX_Y` devices
- Images of a full UBI space, containing several volumes can be created using the `ubinize` utility
  - Can be written to a raw MTD using `nandwrite`



SquashFS

http://squashfs.sourceforge.net/

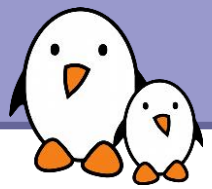
- Filesystem for block storage, so it doesn't support the MTD API.
- However, as it is read-only, it works fine with `mtdblock`, as long as the flash doesn't have any bad blocks
- You can use it for the read-only sections in your filesystem.



SquashFS - How to use

Very simple!

- On your workstation, create your filesystem image:
  `mksquashfs rootdir rootdir.sqfs`
- **Caution**: if the image already exists remove it first, or use the -noappend option.
- Erase your flash partition:
  `flash_eraseall /dev/mtd2`
- Make your filesystem image available to your device (NFS, copy, etc.) and flash your partition:
  `dd if=rootdir.sqfs of=/dev/mtdblock2`
- Mount your filesystem:
  `mount -t squashfs /dev/mtdblock2 /mnt/flash`



Our benchmarks

jffs2

- Dramatically outperformed by ubifs in most aspects.
- Huge mount / boot time unless `CONFIG_SUMMARY` is used.

yaffs2

- Also outperformed by ubifs.
- May not fit all your data
- Ugly file removal time (poor directory update performance?)
- Memory usage not scaling
- ubifs leaves no reason to stick to yaffs2.

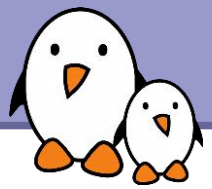
ubifs

- Great performance in all corner cases.

SquashFS

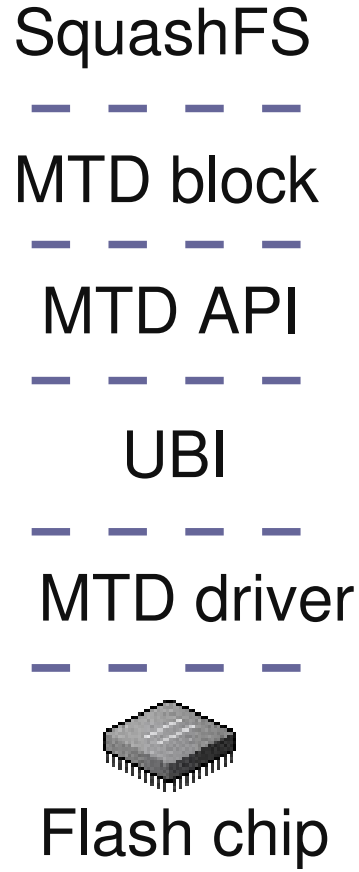
- Best or near best performance in all read-only scenarios.

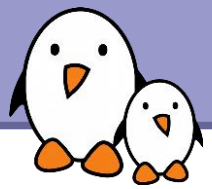
Full benchmark details on
http://free-electrons.com/pub/conferences/2008/elce/flash-filesystems.pdf



Conclusions

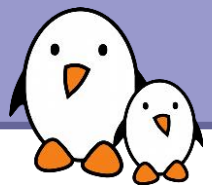
- Convert your jffs2 partitions to ubifs!
- It may only make sense to keep jffs2 for MTD partitions smaller than 10 MB, in case size is critical.
- No reason left to use yaffs2 instead of jffs2?
- You may also use SquashFS to squeeze more stuff on your flash storage. Advisable to use it on top of UBI, to let all flash sectors participate to wear leveling.





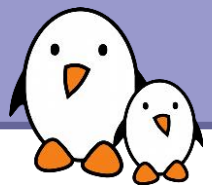
Issues with flash-based block storage

- Flash storage made available only through a block interface.
- Hence, no way to access a low level flash interface and use the Linux filesystems doing wear leveling.
- No details about the layer (Flash Translation Layer) they use. Details are kept as trade secrets, and may hide poor implementations.
- Hence, it is highly recommended to limit the number of writes to these devices.



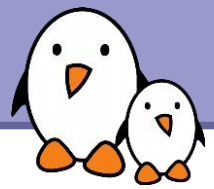
Reducing the number of writes

- Of course, do not use your flash storage as swap area (rare in embedded systems anyway)
- Mount your filesystems as read-only, or use read-only filesystems (SquashFS), whenever possible.
- Keep volatile files in RAM (tmpfs)
- Use the `noatime` mount option, to avoid updating the filesystem every time you access a file. Or at least, if you need to know whether files were read after their last change, use the `relatime` option (default setting since Linux 2.6.30).
- Don't use the `sync` mount option (commits writes immediately). Use the `fsync()` system call for per-file synchronization.
- You may decide to do without journaled filesystems. They cause more writes, but are also much more power down resistant (trade-off).

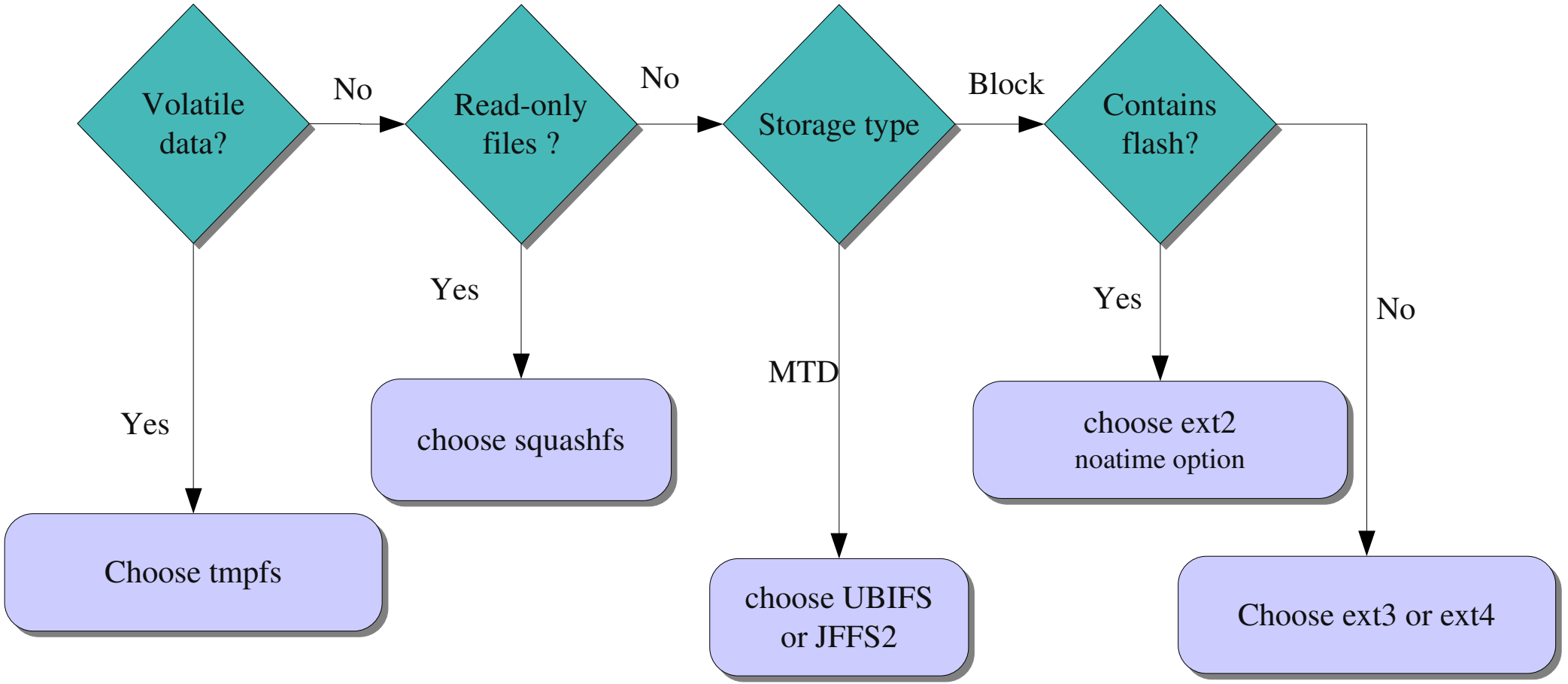


Useful reading

- Introduction to JFFS2 and LogFS:
  http://lwn.net/Articles/234441/
- Nice UBI presentation from Toshiba:
  http://free-electrons.com/redirect/celf-ubi.html
- Documentation on the linux-mtd website:
  http://www.linux-mtd.infradead.org/



Filesystem choice summary

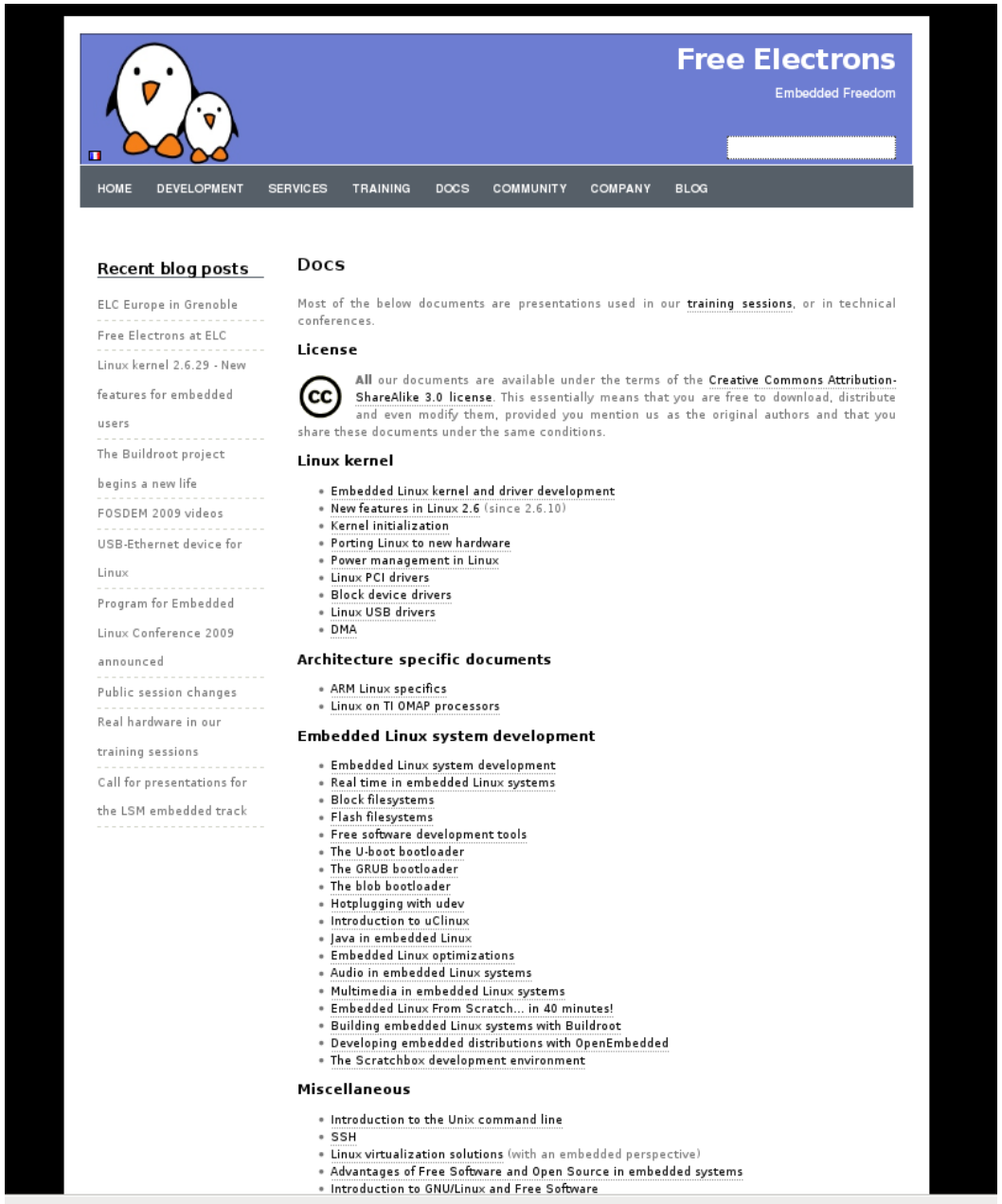


See `Documentation/filesystems/` in kernel sources for details about all available filesystems.

Practical lab – Flash filesysetms

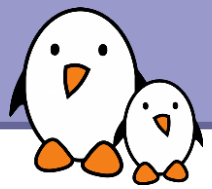
- Creating partitions in your internal flash storage.
- Formating the main partition with SquashFS on mtdblock.
- Using jffs2 for system data.

Related documents



All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs