[Software Development]
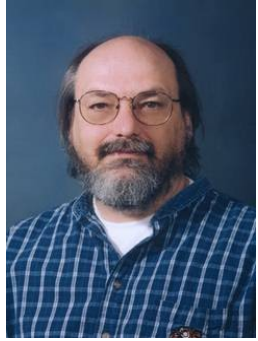
Linux Filesystem

Davide Balzarotti

Eurecom – Sophia Antipolis, France

Previously... on SoftDev

Bill Joy puts together the
first BSD

Thomson wrote the first
version of Unix

Linus Torvalds releases a free
Unix-like Kernel

1965   1970   1975   1980   1985   1990

The Multics Project

Unix is re-written in C

Stallman starts
the GNU project

3

The Unix Philosophy

- *Write programs that do one thing and do it well.*

- *Write programs to work together.*

- *Write programs to handle text streams, because that is a universal interface.*

The Unix Philosophy

- *Write programs that do one thing and do it well.*

- *Write programs to work together.*

- *Write programs to handle text streams, because that is a universal interface.*

The Shell

- Variables

- Shell expansions

- Input/Output and Redirections

- Combining commands
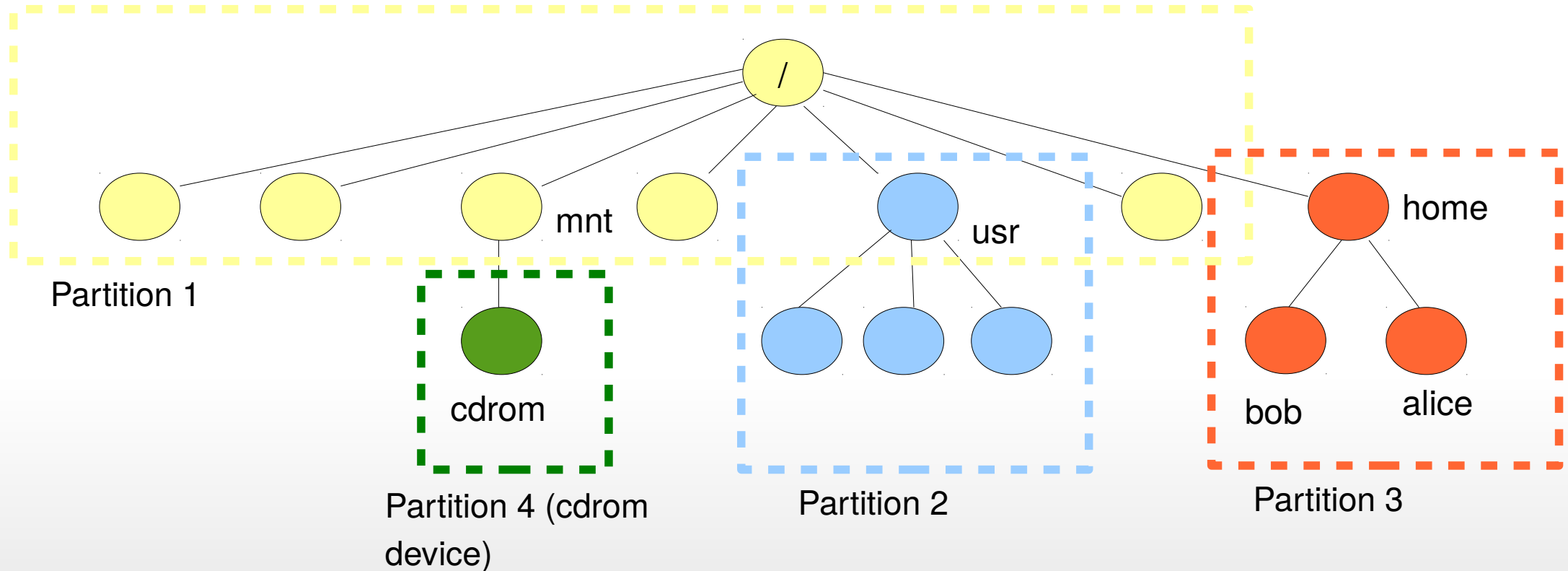
Unix File System

- Everything is a file

    - Directory are files

    - Devices (keyboard, hard disks, printers, ...) are files

    - Why?
      Because any program that can read a document, can also read from a device

- No letters to identify the different partitions

    - Everything is mounted in the same directory tree

    - In Windows, partitions are detected at boot and assigned a drive letter.
      In Linux it is the user that has to explicitly mount each partition

    - The root of the directory tree is /

- File and directory names are case sensitive

Many Partitions under the Same Roo[f|t]
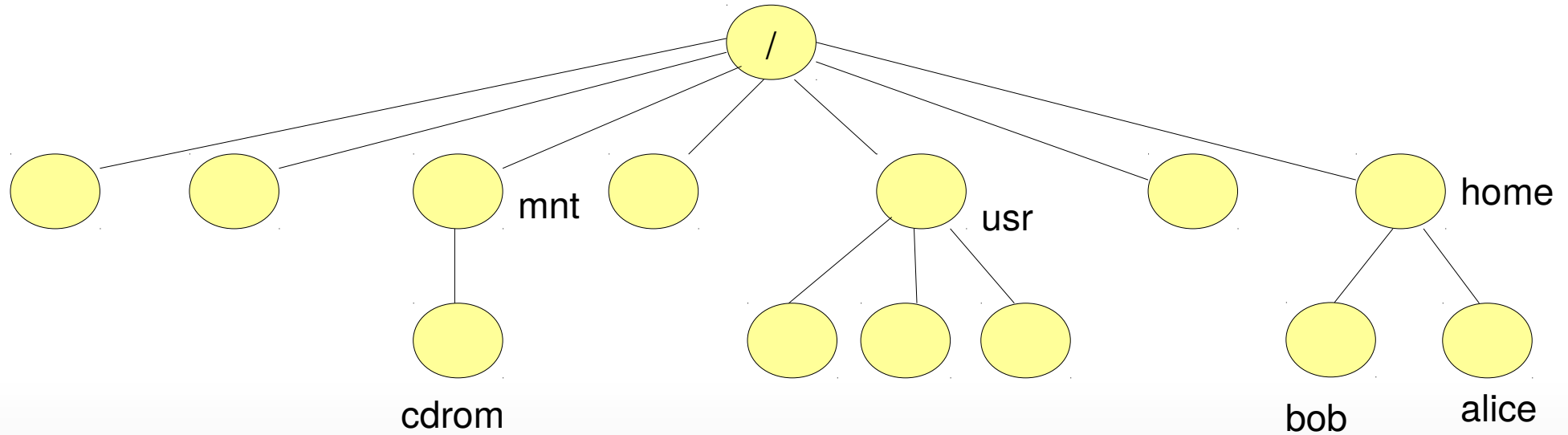
Many Partitions under the Same Roo[f|t]

- Filesystems are mounted using the `mount` and unmounted using the `unmount` commands

Mounting a Filesystem

- Mounting:
  - Usually requires root privileges

    ```
    mount [options] [device] mount-point
    ```
  - Example (to mount a cdrom):

    ```
    mount -t iso9660 /dev/scd0 /mnt/cdrom
    ```

- Partitions that have to be mounted at boot or that can be mounted by normal users can be specified in `/etc/fstab`

```
# Memory stick
/dev/sdb1 /mnt/pen  auto  user,noauto,exec  0   0
```

Device

Mount Point

Filesystem

Options

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```
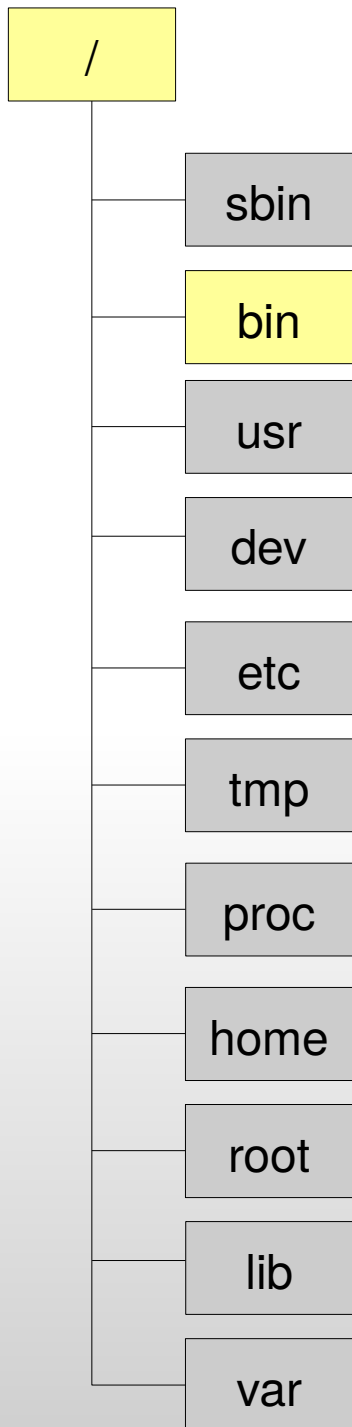
It contains all the programs that are essential to the working of the system. These include system administration as well as maintenance tools.
The programs in this directory require root privileges to be executed

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```

It contains essential commands that can also be run by normal users. For instance the shells and the commands to manipulate files (`cp, mv, cat, ls`..) are usually stored in `/bin/`

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```
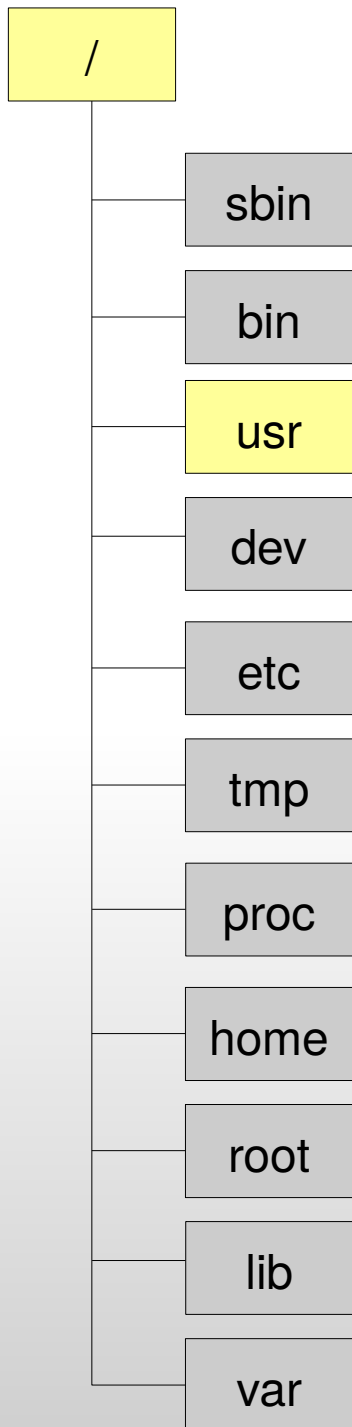
It is usually one of the largest directory in the system (often stored in a different partition).

It contains most of the user (not critical to the system) applications:

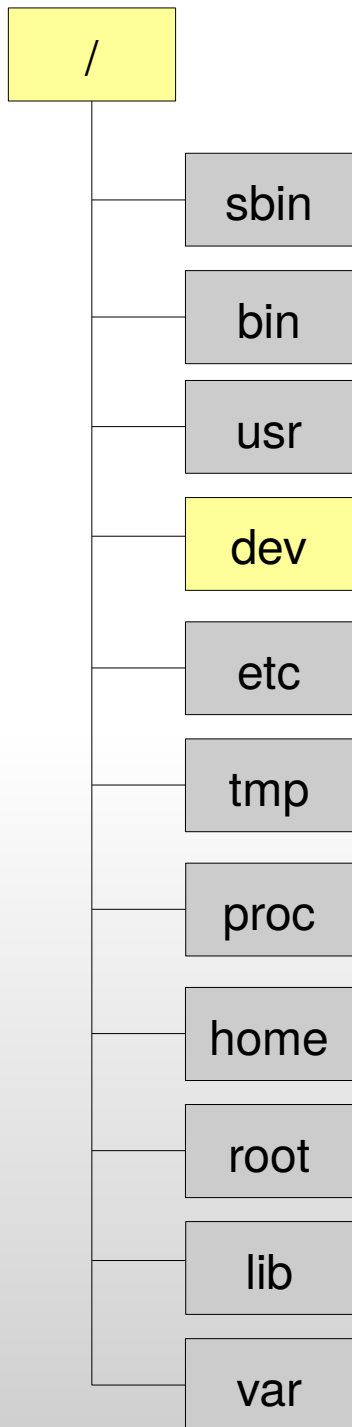`/usr/bin,/usr/sbin,/usr/lib/` – non-critical binaries and libraries

`/usr/include` – headers file for the C compiler

`/usr/doc` – application's documentation

`/usr/src` – source file for the kernel and other applications

`/usr/local/` – apps compiled and installed by the admin locally

`/usr/share/` – arch-independent read-only files

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```
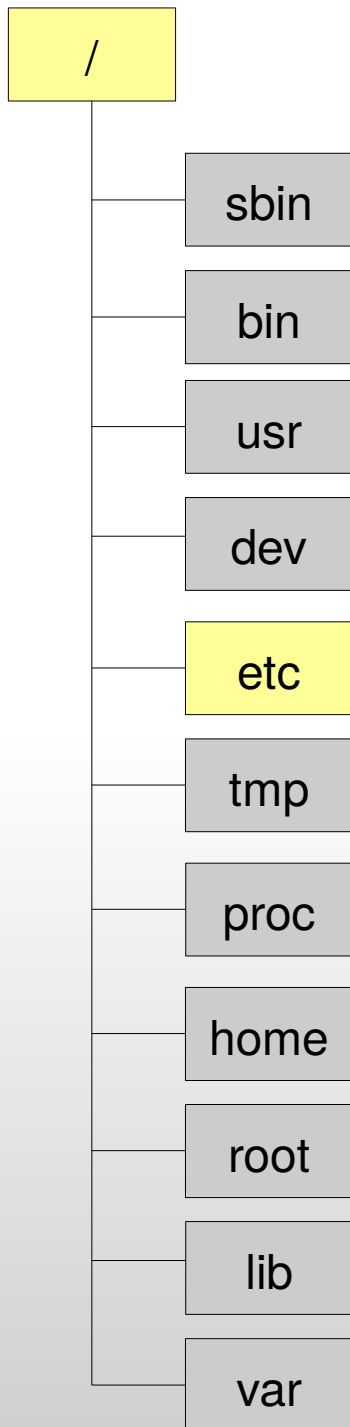
It contains the devices available to the system.
For example:

`/dev/cdrom` – cdrom

`/dev/hda` – the first IDE disk

`/dev/dsp` – the computer speaker

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc ──────── It contains the configuration files (in plain
│               text) for the entire system.
│
│               Configuration files specific for a particular
│               user are kept in the user's home directory
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```

It contains the configuration files (in plain text) for the entire system.

Configuration files specific for a particular user are kept in the user's home directory

14

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```
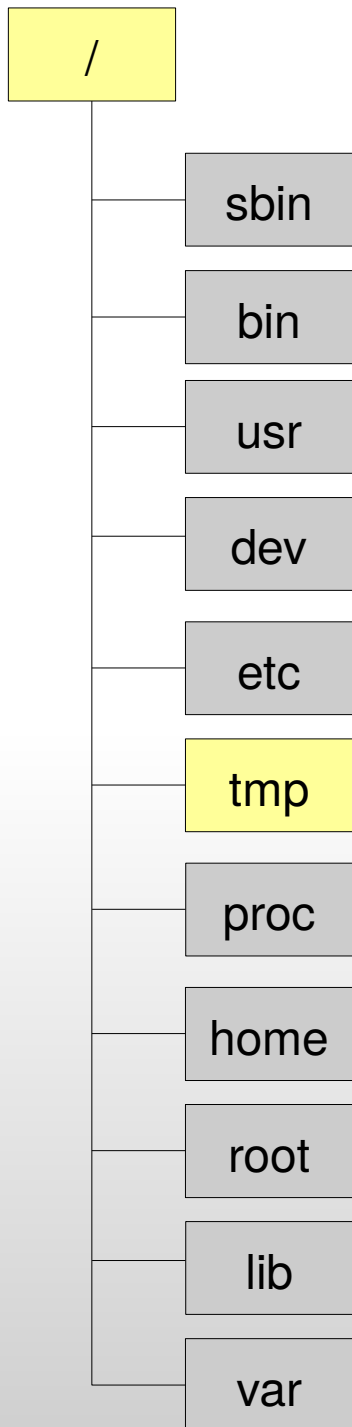
It contains temporary files which are usually deleted at system reboot.
Many programs use this directory to create lock files and for temporary data storage
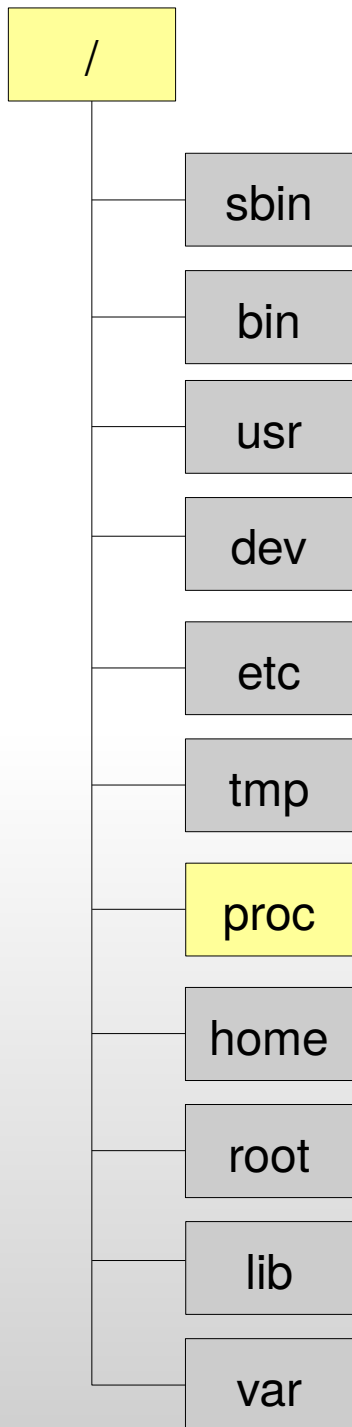
```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc ── It is a "virtual" filesystem that resides in memory and
│           contains files (some readonly, some read-write) that
│           represent information about the Linux kernel and all the
│           running processes
├── home
├── root
├── lib
└── var
```

It is a "virtual" filesystem that resides in memory and contains files (some readonly, some read-write) that represent information about the Linux kernel and all the running processes
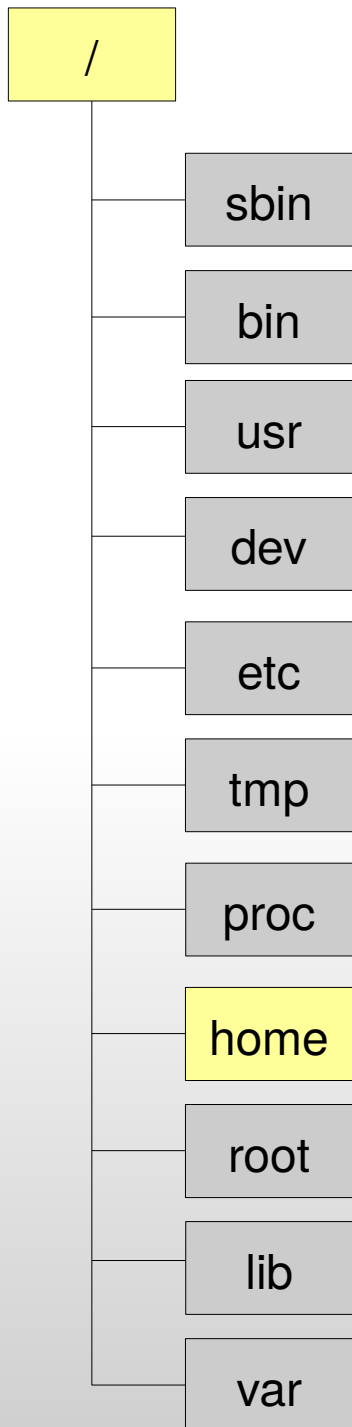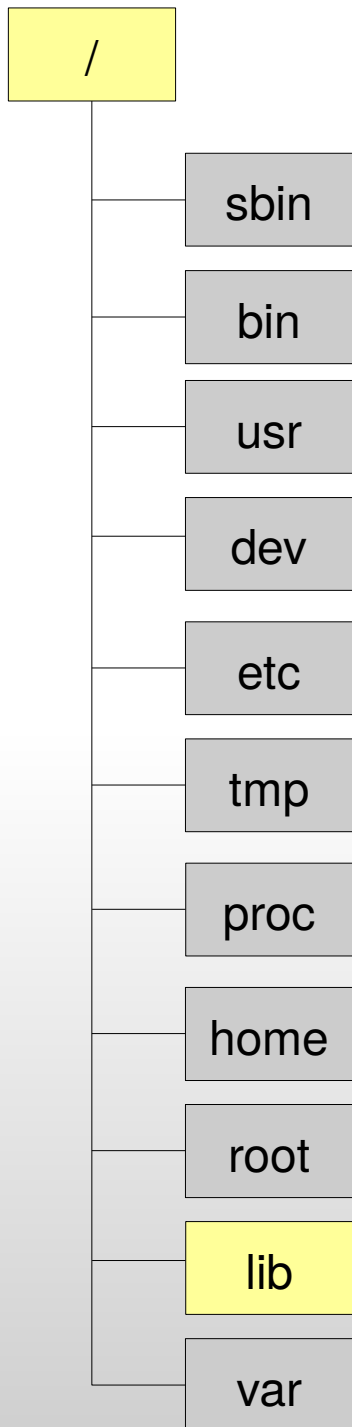
It contains the user home directories.
Each user has his own directory that is accessible only to him (and the administrator).
File starting with a dot (.) in the home usually contains user-specific configurations

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```
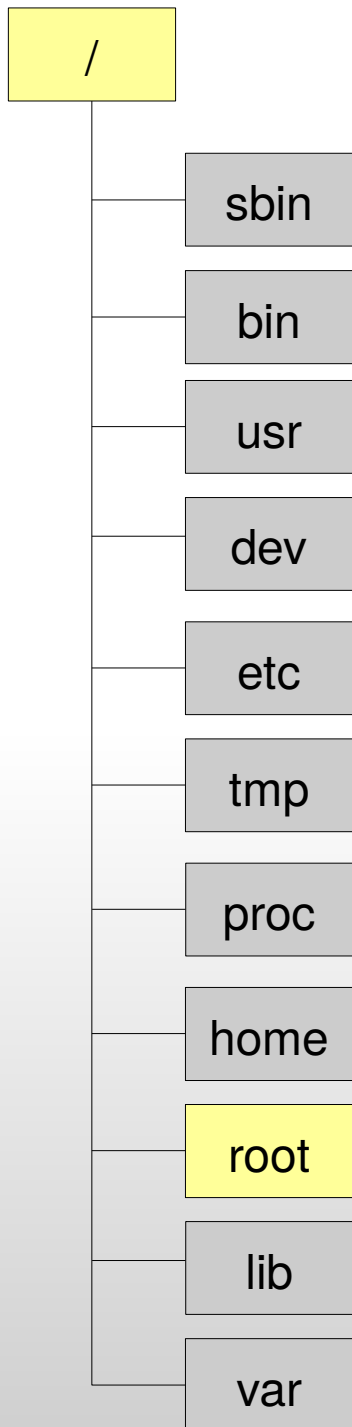
The home directory of the root user

(it is not under `/home` because `/home` is often located on a different partition or even on another system)

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```

Contains the shared library needed to run the binaries in the `/sbin` and `/bin/` directories.

It also contains (under `/lib/modules/`) the kernel modules

19

```
/
├── sbin
├── bin
├── usr
├── dev
├── etc
├── tmp
├── proc
├── home
├── root
├── lib
└── var
```
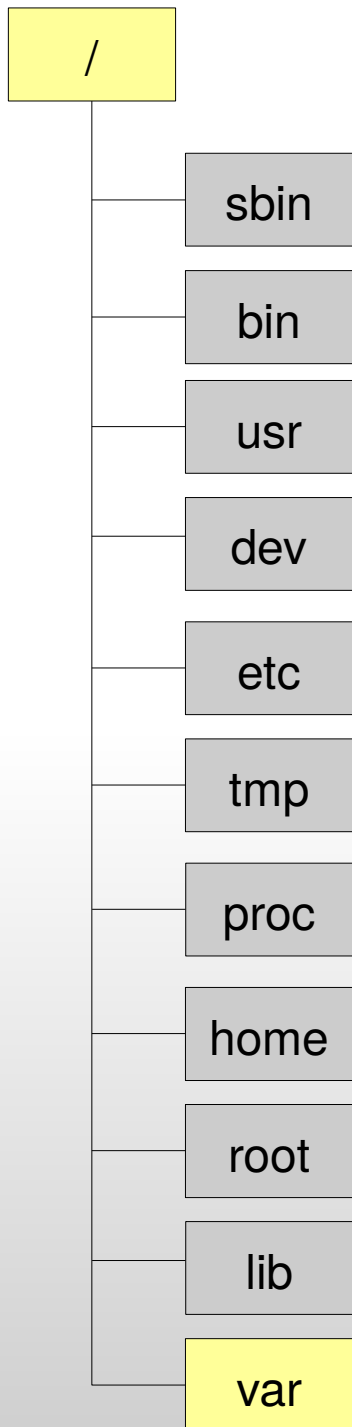
It contains variable data that the system must be able to write during its normal operation

(it is not under `/usr` because `/usr` can sometimes be mounted read-only).

For example, it contains log files, mail and printer spools

20

Moving Around the Filesystem

|	
Print the current directory	`pwd`
Show the file(s) in a directory	`ls [dir/file]`
Move to a certain directory	`cd dir`
Move to the parent directory	`cd ..`
Create a directory	`mkdir dir`
Delete a file	`rm file`
Delete a directory (recursively)	`rm -rf dir`
Copy filea to fileb	`cp filea fileb`
Move filea to path	`mv filea path`
Rename filea to fileb	`mv filea fileb`

File Names

- File names can contain almost any character

  - But it is better to limit your imagination to alphanumeric characters + the underscore (<u>white spaces</u> are very annoying!)

- File extensions can be used to give the user a hint on the file type

  - They are not mandatory and they can be arbitrarily long

  - Unlike Windows, Unix does not use them to identify the file type

  - The fact that a file is executable depends only on its <u>permissions</u>

- The type of the file, depends on its content

  - The command `file` tries to match the file content against a list of signatures to identify its type

```
> file review.txt
review.txt: ASCII English text
```

File Permissions

- Specify the permissions to read/write/execute data for the owner, the owning group, and the rest of the system users

- For a directory

  - read permission: the contents of the directory can be listed

  - write permission: entries in the directory may be added or deleted

  - execute permission: it is possible to access the directory

File Type   Group Members Permissions

d	r	w	x	r	-	x	r	-	x

Owner Permissions   Other Users Permissions
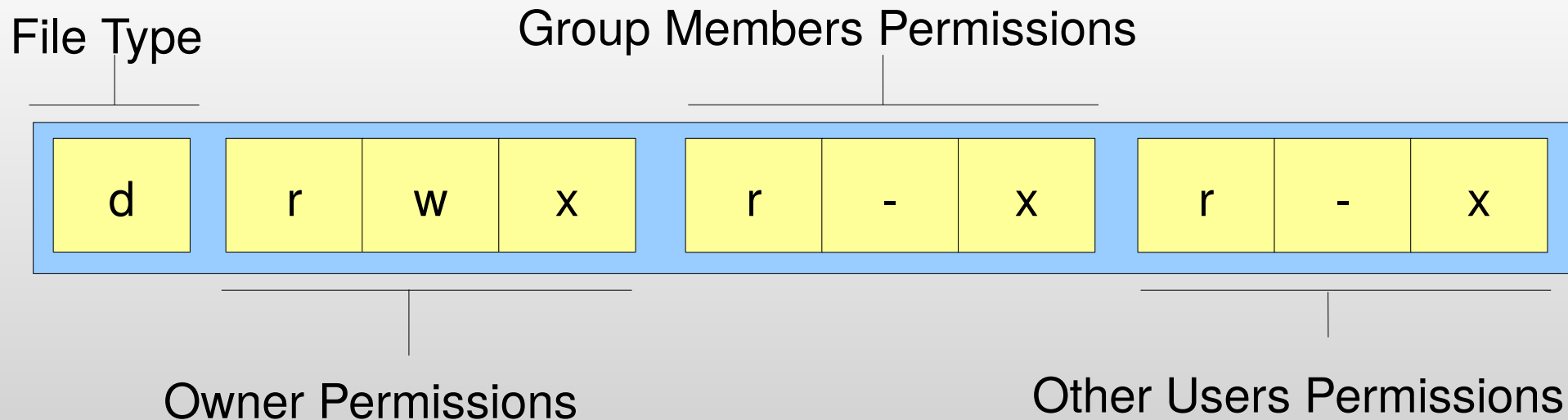
Three More Bits

- SUID or setuid - change user ID on execution.

  - If set, when the file is executed the process will have the same rights as the file owner

- SGID or setgid - change group ID on execution.

  - Same as above, but inherits rights of the group of the owner

  - ! For a directory, it means that when a new file is created in the directory it will inherit the group of the directory (instead of the group of the user who created the file)

- Sticky bit

  - Original purpose is now obsolete

  - For directories, it prevents unprivileged users from removing or renaming a file in the directory unless they own the file or the directory

Links

- Conventional links (hard links)

    - Used to associate several names to the same file

    - Any operation on the file can be done on any of its link

    - The file system maintains a count of the links assigned to each file

        - When the number becomes zero, the file contents are effectively deleted, and the space on the disk is made free

    - But...

        - It is impossible to create links to directories

        - It is impossible to create links between files stored on different file systems

Links

- Symbolic links (soft links)

  - Used to create pointers to other files

  - They can point to any type of file (even non-existent ones)

  - A symbolic link is a special type of file whose contents specify the path of the target file

! If you delete the file, the link remains dangling

! If you delete the link, the file is not affected

Other Useful Commands

Create an hard link X to the file Y	`ln Y X`
Create a soft link X to the file Y	`ln -s Y X`

Change file permissions	`chmod MODE FILE`
	
	`MODE = [ugoa][+-=][strwx]`

Change the file owner	`chown user:group FILE`
Report file systems free space	`df -h`
Calculate directory size	`du -sh DIR`

Change the File Ownership

- In earlier versions of Unix, users were allowed to change the owner and group of their own files

- Since this behavior creates A LOT of problems, BSD restricted the policy to:

    - Only ROOT can change the owner of a file

    - The owner can change the group of a file only to one of the other groups he belongs to

- Changing the file owner usually resets the `set-uid/gid/sticky` bits

- Copying a file change the file attributes (timestamps, ownership, permissions...)

    - `-p` option is used to preserve the attributes (when possible)

File Times (MAC)

- **m**`time`

  - The date of the last modification of the file contents

  - It is updated at each writing of data to the file

- **a**`time`

  - The date of the most recent access

  - It is update at each access (reading or writing) to the file

- **c**`time`

  - The date of the last <u>change</u> to the file (ctime)

  - It is updated at each modification of the file (for instance in the file permissions)

- IMPORTANT: Linux (and Unix) did not originally store the file creation time !! This information is now available on recent filesystems

File Info

```
balzarot > ls -l review.txt
-rw-r--r-- 1 balzarot balzarot 2810 2009-08-05 03:21 review.txt
```

Permissions

Owner (user and group)

Date of last modification (mtime)

Number of hard links

File size

```
balzarot > stat review.txt
  File: `review.txt'
  Size: 2810          Blocks: 8       IO Block: 4096   regular file
Device: 803h/2051d    Inode: 411473   Links: 1
Access: (0644/-rw-r--r--)  Uid: (1000/balzarot) Gid: (1000/balzarot)
Access: 2009-10-09 18:34:50.000000000 +0200
Modify: 2009-08-05 03:21:02.000000000 +0200
Change: 2009-08-05 03:21:02.000000000 +0200
```

The atime Problem

- Since `atime` must be updated every time a file is read...
  so the system has to perform a write for each read operation

  - Kernel developer Ingo Molnár called `atime`

    "*perhaps the most stupid Unix design idea of all times*" and said that "*atime updates are by far the biggest IO performance deficiency that Linux has today*"

- The `atime` behavior can be modified changing the `mount` options

- Starting from the kernel 2.6.30 (middle 2009), the default Linux behavior is to update `atime` <u>only</u> if at least one of the following conditions is true:

  - `atime < mtime`

  - `atime < ctime`

  - `atime` is 24h older