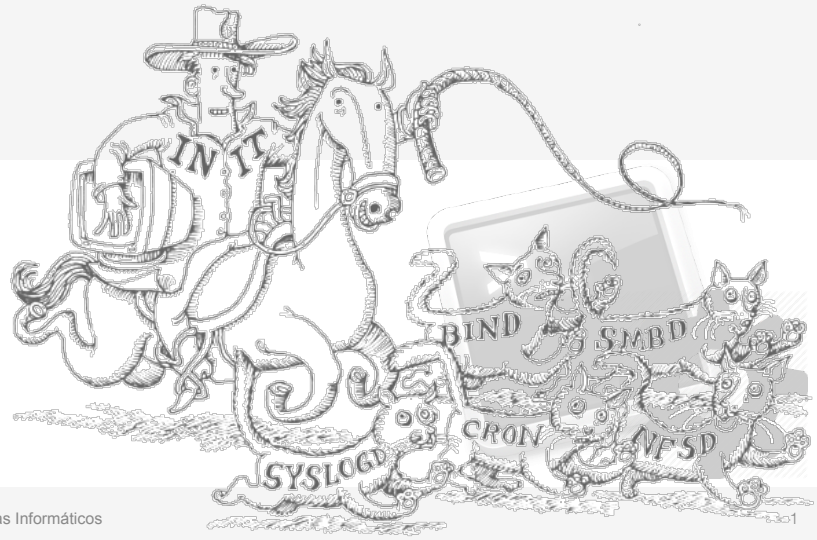
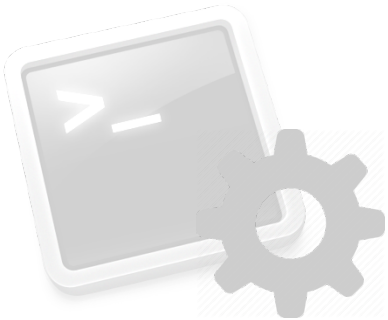
Booting & Shutting Down



Index

- Introduction
- Booting, Stage 1: Hardware
- Booting, Stage 2: Bootloader
  - LILO
  - GRUB
- Booting, Stage 3: Kernel
- Booting, Stage 4: INIT
- Shutting Down

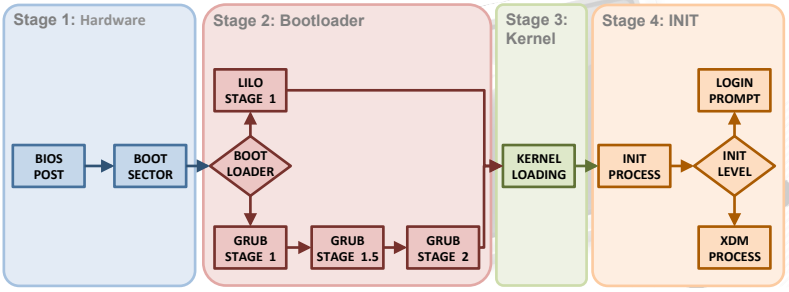


Introduction

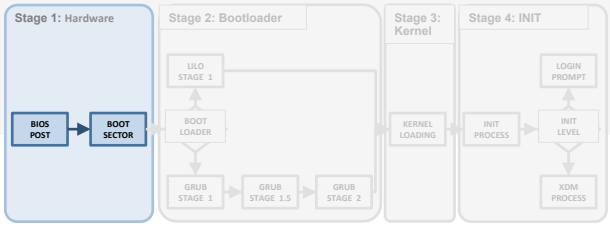
- Booting/Shutting Down are complex procedures, but system provides mechanisms to deal with them.
- … However, this is one of the potential troubles of administration.
- Goals of this Chapter:
  - Understand the basic operation of both procedures.
  - Being able to customize them.
  - Being able to solve generic problems related to Boot process.
- **Bootstrapping**, Where does the name come from?
  - Allusion to “Baron Münchausen”.
  - Defines a process where a simple system starts up another one with higher complexity (starting the system form a small portion of the system itself).

Introduction

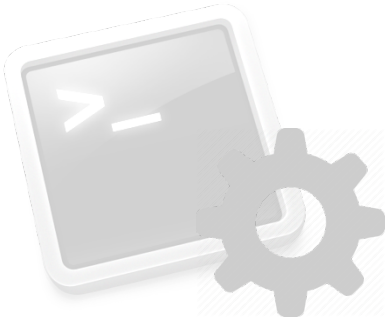
- The main target of Booting process is loading kernel in memory and starting its execution.
  - ¿What’s the content of memory before booting?
  - ¿Where is the kernel before booting?
- It is a sequential process divided in 4 stages:



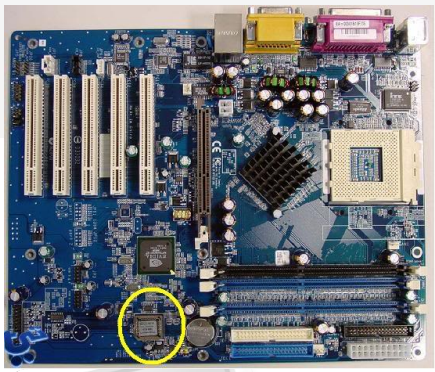
Index



- Introduction
- **Booting, Stage 1: Hardware**
- Booting, Stage 2: Bootloader
  - LILO
  - GRUB
- Booting, Stage 3: Kernel
- Booting, Stage 4: INIT
- Shutting Down

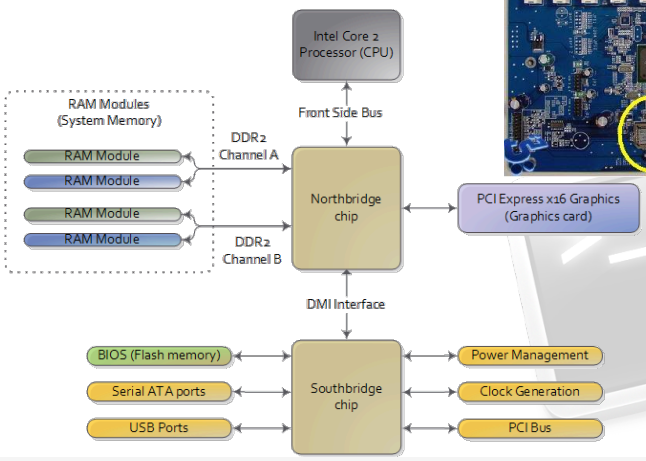


Stage 1: Hardware



- First Steps:
  - After pushing Power-On button, el **Reset Vector** indicates the CPU the direction of the first instruction to execute (FFFFFFF0h for x86).
  - Such direction corresponds to an EPROM/Flash (motherboard) that stores the code corresponding to the Firmware (memory-mapped I/O).
- Firmware:
  - Stores Hardware configuration for the system.
  - Some configuration parameters with its own power supply (battery).

Stage 1: Hardware

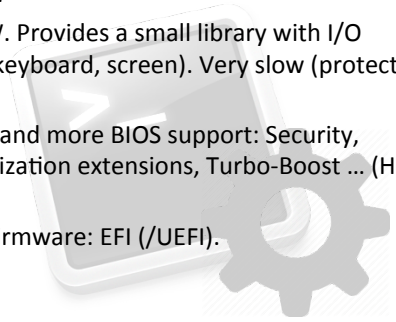


Stage 1: Hardware

- Tasks to perform:
  - **Power-on-self-test** (POST): examination, verification and start up of hardware devices (CPU, RAM, Controllers, etc.)
  - Configuration of previous aspects, independent of OS (Virtualization extensions, security, etc.)
  - Start up the Operating System: In the case of BIOS, look for the OS loader in the first block (512 bytes) [**Master Boot Record (MBR)**] from the booting device in the configured order. When found, the content is loaded in memory.
- Two main kinds of Firmware:
  - BIOS: Basic Input Output System
  - EFI: Extensible Firmware Interface

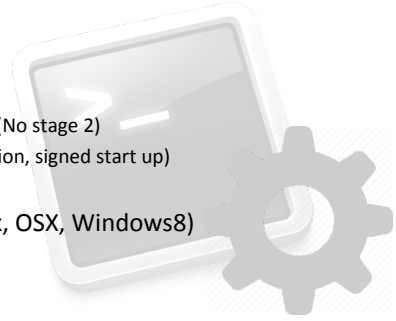
Stage 1: Hardware

- **BIOS** (Basic Input/Output System):
  - 1975: First appearance in the Operating System CP/M.
  - It runs in real address mode (16 bit): 1MB of addressable memory.
  - 1990: appears "BIOS setup utility": allows the user to define some configuration options (boot priority).
  - ROM customized for a particular HW. Provides a small library with I/O functions to work with peripherals (keyboard, screen). Very slow (protected to real mode).
  - Emerging applications require more and more BIOS support: Security, Temperature/Power metrics, Virtualization extensions, Turbo-Boost … (Hard to put all that in 1MB).
  - 2002: Intel develops an alternative firmware: EFI (/UEFI).

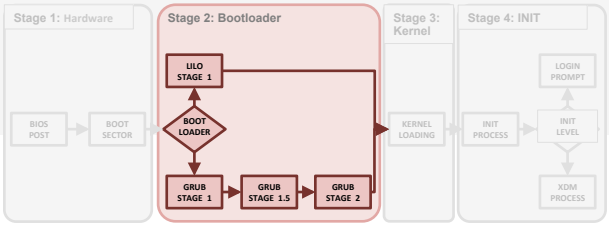


Stage 1: Hardware

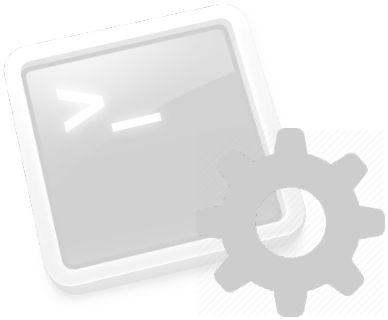
- **EFI/UEFI** (Unified Extensible Firmware Interface):
  - 2002: Itanium platform from intel provides EFI firmware.
  - 2005: UEFI. Consortium of companies takes control over the firmware. Unified EFI Forum.
  - Works in 32/64 bits mode.
  - Much more flexible than BIOS
    - Supports big disks (<1TB)
    - Supports more booting devices (red)
    - Can eliminate the need of a bootloader (No stage 2)
    - Improved Security (network authentication, signed start up)
    - Can be modified in-flight.
  - Requires support from the OS (Linux, OSX, Windows8)
    - Windows XP in the first intel iMacs??
  - Can emulate BIOS.
  - Supported for VirtualBox



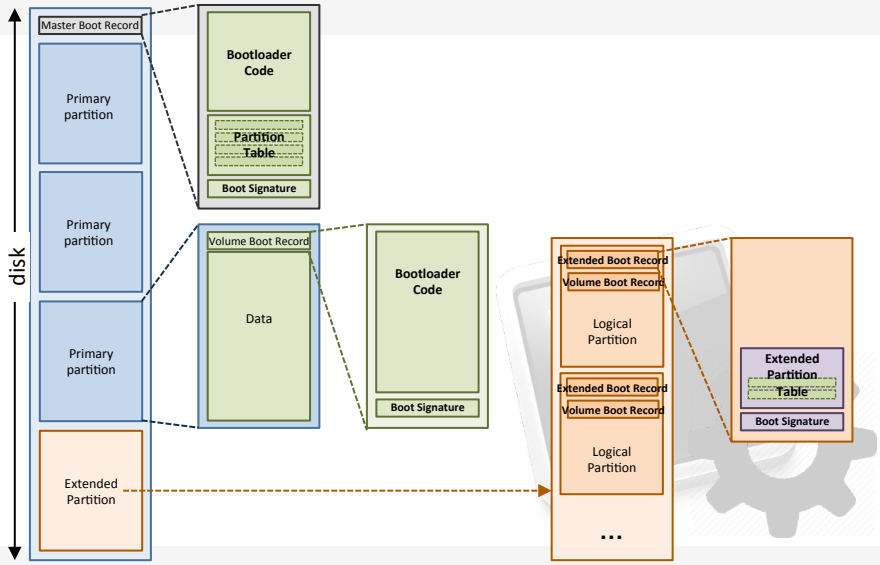
Index



- Introduction
- Booting, Stage 1: Hardware
- **Booting, Stage 2: Bootloader**
  - **LILO**
  - **GRUB**
- Booting, Stage 3: Kernel
- Booting, Stage 4: INIT
- Shutting Down



Previous: Disks & Partitions



Previous: Disks & Partitions

- **Master Boot Record (MBR)**:
  - First block of the Disk, 512 Bytes.
  - **Partition Table**: information about four primary partitions: begin and end blocks, size, etc. (64 bytes)
  - **Boot Signature**: Numerical value indicating the presence of valid bootloader code in the code field (0x55AA) (2 bytes).
- **Volume Boot Record (VBR)**:
  - First block of each primary partition.
  - Could contain bootloader code (indicated by Boot Signature).
- **Extended Partition**:
  - Partition that can be sub-divided into multiple **logical partitions**.
  - **Extended Boot Record (EBR)**: First block of each logical partition. It only contains a partition table with two fields. **Extended partition table** forms a linked list with all logical partitions.

Previous: Disks & Partitions

- **Linux Naming Convention:**
  - Remember: I/O devices are treated as files. Under directory /dev we find all system disks.
  - generic PC: 2 IDE controllers, each can have two devices (master/slave).
    - /dev/**hda**: first device (master) of the first IDE controller.
    - /dev/**hdb**: second device (slave) of the first IDE controller.
    - /dev/**hdc**: first device of the second controller.
    - /dev/**hdd**: second device of the second controller.
  - In a disk, each **primary partition** is identified with a number from 1 to 4.
    - /dev/**hda1**: first primary partition of the hda disk.
  - **Lógical partitions** start from 5.
    - /dev/**hda5**: first logical partition of hda disk.
  - In **SCSI devices** same naming convention, changing “sd” by “hd”
    - /dev/**sda1**

Stage 2: Bootloader

- Hardware requires an OS in charge of providing all the functionality in a computer.
- Target: loading in memory OS kernel and start running it. Loader with different locations: USB, CD, Disk …
- **Stage 1**:
  - Located in **MBR**: 512 first bytes (block 0) of the **active device**.
  - Loaded in memory by BIOS (Stage 1).
  - Triggers, when executed, the load and execution of Stage 2.
- **Stage 2**:
  - Located in the **active partition**, where the kernel is placed.
  - Loads the kernel in memory and transfers control to it (data initialization, drivers, check CPU, etc.)
  - After this process, the **init** process is executed (Stage 3)

Stage 2: LILO

- **LI**nux **Lo**ader:
  - Two stage Bootloader.
  - Does not "understand" about operating system, neither about file system. Only works with physical locations.
  - Obsolete (but easy to follow for academic purpose)
- Steps:
  - Master boot loads LILO from the first active partition and runs it.
    - LILO can be in the MBR or in the Boot Block of a primary partition. In the second case, MBR contains the necessary code to load LILO from another block.
  - LILO requests the user the kind of boot wanted (partition, kernel, mode). Through a prompt.
  - LILO loads the kernel and a ramdisk.
  - The kernel starts running once it is loaded into memory.

Stage 2: LILO

- Configuration: /etc/**lilo.conf**

```
boot=/dev/hda #o by ID
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.6.2-2
        label=linux
        read-only
        root=/dev/hda2 #o by UUID
        initrd=/boot/initrd-2.4.2-2.img

other=/dev/hda1
        label=dos
        optional
```

Device where LILO is installed (IDE/SATA/ Floppy…)

File with information about disk blocks with the files required to boot system.

Loader Assembly code.

Kernel for booting and its options

Linux system partition (/). Not necessarily a disk (usb loader).

Filesystem loaded in memory as a ramdisk. Software support not provided by the kernel to initialize the system.

Link to other loader (boot a different OS)

Stage 2: LILO

- Configuration: /etc/**lilo.conf**
  - Any change in the files employed in boot process (boot.b, kernel, ramdisk) requires loader update:
    - map file must reflect those changes, otherwise booting process is corrupted.
    - Check if map file is updated: # lilo -q
    - Update map file: # lilo [-v]
- A booting error cannot be fixed from the shell…
- Possible error sources:
  - Installation of a new OS overwriting MBR (M$)
  - Failed kernel compilation
  - Modification in boot files without map updating.
- Rescue Systems:
  - mkbootdisk
  - Installation Live CD (option rescue) or specialized (SystemRescueCD)

Stage 2: GRUB/GRUB2

- **GR**and **U**nified **B**ootloader: linux loader
  - Bootloader with three stages.
  - Can work with file systems (ext2, ext3, ext4,…), directly accessing partitions (no map files)
  - Supports EFI
  - Much more flexible, has its own mini-shell (grub>)
    - Booting parameters can be decided through that prompt. It is possible to indicate the kernel and the ramdisk before startup (booting a OS which was not in the boot menu).
    - "c" from the startup window opens the console with the values for the selected input.
    - "e" edits each input in n-curses format.
    - "kernel", "initrd" loads a kernel or a ramdisk.
    - "boot" boots your OS
    - Access to the file system and command has auto-complete (TAB)
  - Currently the most employed is GRUB2.

Stage 2: GRUB/GRUB2

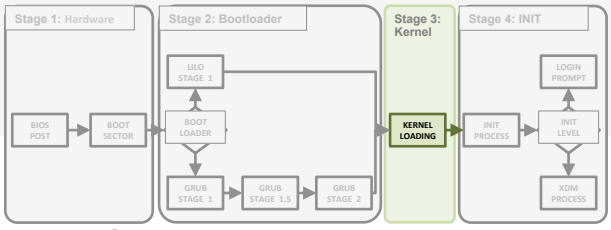
- **GR**and **U**nified **B**ootloader:
  - Configuration:
    - More complex scripts than LILO. Advantage: Modifications in files required to boot (kernel o initrd) are processed "automatically".
    - Everything in /etc/default/grub and /etc/grub.d/
    - Final configuration (/boot/grub) is performed through the command "update-grub".
  - Stages:
    - Stage 1: boot.img stored in MBR (or VBR), loaded in memory and executed (loads the first sector of core.img).
    - Stage 1.5: core.img stored in the blocks between MBR and first partition (MBR gap), loaded in memory and executed. Loads its configuration file and drivers for the file system.
    - Stage 2: Load Kernel and ramdisk, accessing directly to the file system (/boot/grub).

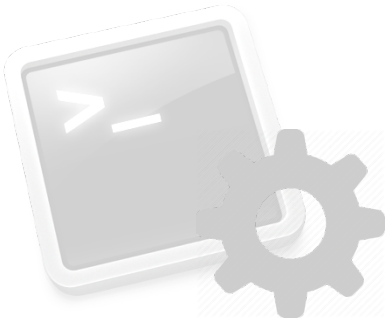
Stage 2: Bootloader

- Having physical access to a system, stages 1 & 2 can become a weakness.
  - Modifying boot options we could obtain superuser privileges.
- Protect BIOS and loader with password.
- Example: Protection of GRUB2 with password
  - Edit /etc/grub.d/00_header and at the end of the file add (remember to perform update-grub after that):

```
cat << EOF
set superusers="alumno"
password alumno <<<<<secuencia de grub-mkpasswd-pbkdf2>>> o <<password-plano>>
export superusers
EOF
```

Index

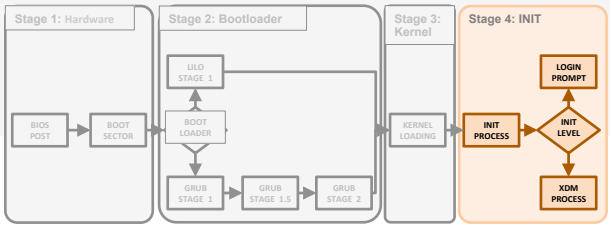




Stage 3: Loading the Kernel

- The bootloader has loaded in memory kernel & ramdisk files
  - vmlinuz-2.6.26-2-686
  - initrd.img-2.6.26-2-686
- Once finalized Stage 2, kernel execution starts:
  - The Kernel **un-compresses** itself.
  - Detects memory map, the **CPU** and its features supported.
  - Starts the **display** (console) to show information through the screen.
  - Checks the **PCI bus**, creating a table with the peripheral detected.
  - Initializes the system in charge of **virtual memory** management, including swapper.
  - Initializes the **drivers** for the peripherals detected (Monolithic or modular).
  - Mount **file system** root ("/").
  - Calls the **init** process (Stage 4): PID 1, father of the rest of processes.

Index



Stage 4: INIT

- The init process performs the following tasks:
  - Step 1: **Configuration**: read from the file **/etc/inittab** the initial configuration of the system: Operation mode, runlevels, consoles,…
  - Step 2: **Initialization**: Runs the command **/etc/init.d/rc.S** (debian), which performs a basic initialization of the system.
  - Step 3: **Services**: According to the runlevel configured, runs the scripts/ services pre-established for that runlevel.
- Runlevels (Operation modes)
  - Standard: 7 levels. Each distribution its own configuration (here Debian)
  - Level **S**: only executed at boot time (replaces /etc/rc.boot)
  - Level 0: **Halt**. Employed to Shut down the system.
  - Level 1: **Single User**. Maintenance tasks (no active network)
  - Level 2-5: **Multiuser**. All the network and Graphical services activated.
  - Level 6: **Reboot**: Similar to level 0.

Stage 4: INIT

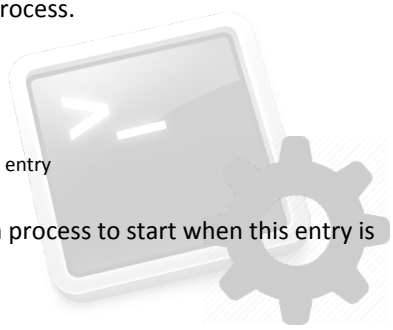
- Step 1, Configuration. The file /etc/inittab:

```
# /etc/inittab: init(8) configuration.

# The default runlevel.
id:2:initdefault:

# Boot-time system configuration/initialization
# script. This is run first except when booting in
# emergency (-b) mode.
si::sysinit:/etc/init.d/rcS

# What to do in single-user mode.
~~:S:wait:/sbin/sulogin

# /etc/init.d executes S and K scripts upon change
# of runlevel.
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6
```

```
# Normally not reached, but fallthrough in case of
# emergency.
z6:6:respawn:/sbin/sulogin

# What to do when CTRL-ALT-DEL is pressed.
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
…

# Note that on most Debian systems tty7 is used by
# the X Window System, so if you want to add more
# getty's go ahead but skip tty7 if you run X.
1:2345:respawn:/sbin/getty 38400 tty1
2:23:respawn:/sbin/getty 38400 tty2
3:23:respawn:/sbin/getty 38400 tty3
4:23:respawn:/sbin/getty 38400 tty4
5:23:respawn:/sbin/getty 38400 tty5
6:23:respawn:/sbin/getty 38400 tty6
```

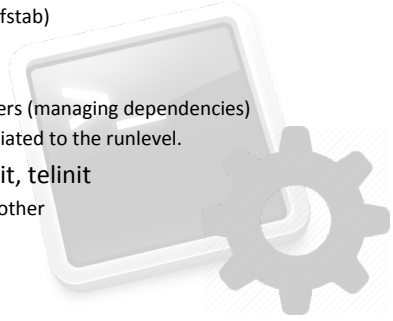
Stage 4: INIT

- Stage 1, Configuration. The file /etc/inittab:
  - Line format: **id:runlevels:action:process**
  - **id:** identifier for the entry inside inittab
  - **runlevels:** execution levels for that entry (empty means all)
  - **action:** What must init do with the process.
    - wait: wait until it finishes
    - off: ignore the entry (deactivated)
    - once: run only once
    - respawn: rerun the process if it dies
    - sysinit: ask the user what to do with that entry
    - Special: ctrlaltdel
  - **process:** sh line indicating init which process to start when this entry is reached.



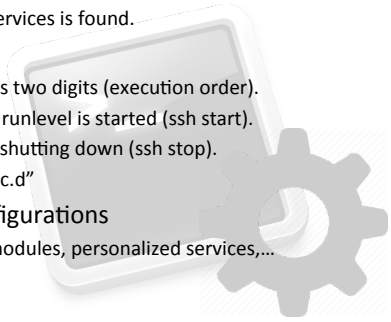
Stage 4: INIT

- Step 2, Initialization. The file /etc/init.d/rc:
  - Input parameters: the runlevel. Example rc 2: multiuser
  - Tasks:
    - Establish PATHs
    - Load swap space: swapon
    - Check and mount local filesystems (/ets/fstab)
    - Activate and configure the network
    - Remove not necessary files (/tmp)
    - Configure tje kernel. Load modules: Drivers (managing dependencies)
    - Triggers the startup of the services associated to the runlevel.
  - Modifying the runlevel: command init, telinit
    - Allows changing from one runlevel to another
    - ¿Single User?
    - Restore original state.



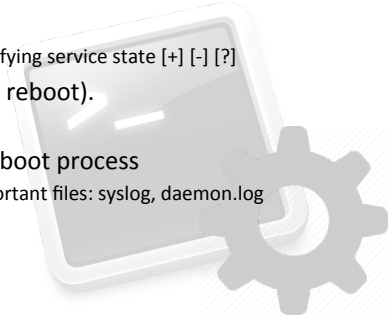
Stage 4: INIT

- Step 3, services. The directories /etc/init.d and /etc/rcN.d:
  - All the services available are found in /etc/init.d
    - Examples: cron, ssh, lpd, …
  - How to indicate each runlevel which ones to start?
    - With a special directory, /etc/rcN.d/ (being N el runlevel).
    - In these directories a list of links to the services is found.
  - The directory /etc/rcN.d/
    - The links begin with letters "S" or "K" plus two digits (execution order).
    - "S": executed in ascending order when a runlevel is started (ssh start).
    - "K": executed in descending order when shutting down (ssh stop).
    - These links are controlled with "update-rc.d"
  - S99local: script to perform local configurations
    - minor booting aspects: auxiliary kernel modules, personalized services,…
    - Employed by the administrator
    - It really runs the script /etc/rc.local



Stage 4: INIT

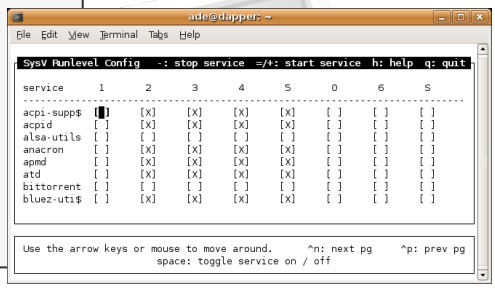
- Manual administration of services:
  - After booting process, services can be modified (stop running services or start new services).
  - Directly through its script (example ssh):
    - # /etc/init.d/ssh [stop/start/restart/status]
  - Or through the command service:
    - service --status-all: reads /etc/init.d/ verifying service state [+] [-] [?]
  - These changes are volatile (lost after reboot).
    - Permanent with update.rc-d
  - Checking possible errors concerning boot process
    - # tail -f /var/log/messages (Another important files: syslog, daemon.log
    - # ls -lart /var/log



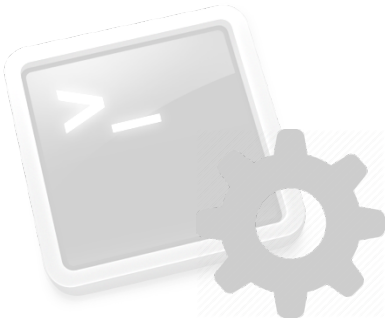
Stage 4: INIT

- Manual administration of services:
  - Examples of start script and services command:

```
#!/bin/sh
#SIMPLIFICADO

[ -f /usr/local/sbin/sshd2 ] || exit 0

PORT=

PORT=`grep Port /etc/ssh2/sshd2_config | awk '{ x = $2 } END {print x}' -`
if [ "X$PORT" = "X" ]
then
        PORT=22
fi

# See how we were called.
case "$1" in
  start) # Start daemons.
        echo -n "Starting sshd2 in port $PORT: "
        /usr/local/sbin/sshd2
        echo "done."
        ;;
  stop) # Stop daemons.
        echo -n "Shutting down sshd2 in port $PORT: "
        kill `cat /var/run/sshd2_$PORT.pid`
        echo "done."
        ;;
  restart)
        $0 stop
        $0 start
        ;;
  *)
        echo "Usage: sshd2 {start|stop|restart}"
        exit 1
esac

exit 0
```

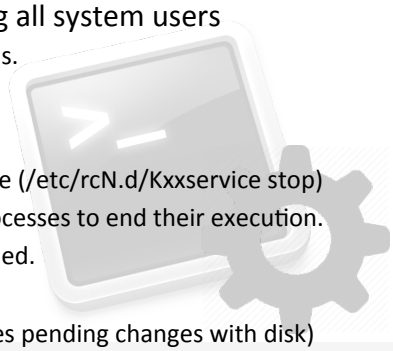


Index



Shutting Down

- Never shut down directly (reset!).
  - If this rule is not respected, there is a high probability of loosing or corrupting system files (with a bit of bad luck, fully broken system)
  - Intermediate Buffers for disk read/write. Synchronization.
- Never shut down without warning all system users
  - Periodically programmed shut-downs.
- Steps for a correct shut down:
  - Warn all the users previously.
  - Stop all the services associated to the (/etc/rcN.d/Kxxservice stop)
  - Send the specific signal to all the processes to end their execution.
  - Users and processes still present, killed.
  - Subsystems shut down sequentially.
  - File System unmounted (synchronizes pending changes with disk)



Shutting Down

- Command **shutdown**:
  - Format: /sbin/shutdown -<options> time message
    - Option -r: reboot instead power off
    - Option -h: stop the system(with ACPI).
  - Message: message sent to all users.
  - time: delay to begin the shutdown (mandatory)
    - Format: hh:mm
    - Supports now+,minutes
- /etc/shutdown.allow or inittab
  - Avoid Ctrl+Alt+Del
- Other commands: /sbin/halt, /sbin/poweroff