

- 101 - Architettura del sistema
 - 101.1 - Determine and configure hardware settings [2]
 - Gli interrupts in Linux
 - ioports e DMA in Linux
 - File /proc/ioports
 - File /proc/dma
 - File /proc/iomem
 - Dispositivi e drivers in Linux
 - Dispositivi ColdPlug e HotPlug
 - La cartella /dev/
 - Attivare, disattivare e visualizzare i driver attivi
 - Configurazione di un modulo
 - La cartella /sys/
 - D-Bus
 - System Bus:
 - Session Bus:
 - Approfondimento: intercettare le comunicazioni su dbus
 - 101.2 - Boot the system [3]
 - BIOS vs UEFI
 - Sequenza di boot su sistemi BIOS
 - Sequenza di boot su sistemi UEFI
 - initrd e initramfs
 - Bootloaders
 - Chain loading
 - Il sistema di init
 - Operazioni di recovery e diagnostica al boot
 - Avvio in modalità rescue
 - Reset della password
 - Consultare opzioni di avvio e configurazione del kernel
 - Diagnosi e troubleshooting
 - Leggere i log con journalctl
 - 101.3 - Change runlevels / boot targets and shutdown or reboot system [3]
 - Cambiare runlevel all'avvio
 - Cambiare runlevel a sistema avviato
 - Cambiare il runlevel predefinito
 - File /etc/inittab (systemV)
 - La cartella /etc/init.d/ (systemV)
 - Comandi shutdown e wall
 - Come funziona systemd
 - Systemd e Systemd User
 - Dove si trovano le systemd unit
 - Com'è fatta una systemd unit
 - Il demone acpid

- 102 - Installazione di Linux e Package Management
 - 102.1 - Design hard disk layout [2]
 - Alterare la tabella delle partizioni
 - Ridurre o espandere una partizione
 - Swap
 - Swappiness
 - Creare un'area di swap
 - Progettare il partizionamento dei dischi
 - Cos'è e come funziona LVM (awareness)
 - LVM: aggiungere e rimuovere dischi
 - La partizione EFI (ESP)
 - 102.2 - Install a boot manager [2]
 - GRUB Legacy (BIOS)
 - Troubleshooting e recovery da GRUB
 - Come funziona GRUB 2
 - Troubleshooting - Menu di GRUB nascosto
 - 102.3 - Manage shared libraries [1]
 - Static vs Dynamic linked libraries
 - Come funziona il linking dinamico
 - La variabile LD_LIBRARY_PATH
 - 102.4 - Use Debian package management [3]
 - dpkg
 - apt-cache (legacy)
 - apt-get (legacy)
 - apt
 - 102.5 - Use RPM and YUM package management [3]
 - rpm
 - yum
 - dnf
 - 102.6 - Linux as a virtualization guest [1]
 - Virtualizzazione vs Containerizzazione
 - IaaS, PaaS, SaaS e terminologia Cloud
 - Clonazione di macchine virtuali e accorgimenti
 - clonazione
 - esportazione
 - cloud-init
- 103 - GNU and Unix Commands.
 - 103.1 - Work on the command line [4]
 - Cambiare shell
 - Entrare e uscire dalla shell
 - Le variabili nella shell bash
 - Altre opzioni della bash
 - Gli ALIAS in bash
 - La bash history

- Informazioni sul kernel: uname
- Ottenere aiuto: man e help
- Quoting in bash
- 103.2 - Process text streams using filters [2]
 - dividere e unire file con split e cat
 - cut
 - head e tail
 - grep
 - tr (o translate)
 - wc
 - nl
 - sort e uniq
 - awk
 - less
 - paste
 - od
 - sed (intro)
 - 1: Sintassi delle espressioni Regolari
 - 2: SED - sostituire una parola
 - 3: SED - i separatori
 - 4: SED - aggiungere apici o parentesi
 - 5: SED - duplicare un match
 - 6: SED - scambiare l'ordine dei match
 - 7: SED - eliminare numeri o testo o cambiarne l'ordine
 - 8: SED - eliminare corrispondenze duplicate
 - 9: SED - tips and tricks
 - zcat, bzcat, xzcat
 - md5sum, sha256sum, sha512sum
- 103.3 - Perform basic file management [4]
 - Copiare, spostare, creare, rinominare file da shell
 - Archiviazione e compressione di file e cartelle
 - rpm, cpio, rpm2cpio
 - Kilobit, Kilobyte e Kibibyte
 - Il leggendario comando dd
 - find, locate, updatedb
 - bunzip2, unxz, gunzip
- 103.4 - streams, redirection e pipe [4]
 - Redirezione dell'output
 - Redirezione dell'input
 - L'operatore pipe
 - Il comando tee
 - Il comando xargs
- 103.5 - Create, monitor and kill processes [4]
 - I jobs in Linux

- Eseguire processi con nohup
- ps e pgrep
 - Il comando ps
 - Il comando 'pgrep'
- POSIX SIGNALS
- kill, pkill e killall
- top, free, uptime
- La cartella proc
- watch : tenere monitorati i comandi
- tmux moltiplicatore di terminali
- 103.6 - Modify process execution priorities [2]
 - nice, renice e priority
- 103.7 - Search text files using regular expressions [3]
 - Le espressioni regolari (regex)
- 103.8 - Basic file editing [3]
 - Introduzione a VI / VIM
 - Default EDITOR e nano
- Appendice: Riassunto comandi più comuni
 - alias e type
 - shell history
 - manuale
 - informazioni sul sistema
 - lavorare con i file di testo
 - gestione dei file
 - streams, redireciton e pipes
 - gestione processi e risorse
- 104 - Dispositivi, Filesystem Linux, Filesystem Hierarchy Standard
 - 104.1 - Create partitions and filesystems [2]
 - MBR vs GPT
 - FAT32 vs exFAT ; Btrfs
 - 104.2 - Maintain the integrity of filesystems [2]
 - Blocchi riservati
 - debugfs (rimosso in 101-500)
 - dumpe2fs (rimosso in 101-500)
 - 104.3 - Control mounting and unmounting of filesystems [3]
 - Montaggio e punto di mount
 - Il file fstab
 - lsblk e blkid
 - systemd mount units
 - 104.5 - Manage file permissions and ownership [3]
 - leggere i permessi
 - Cambiare i permessi - chmod
 - sticky-bit
 - set-UID e set-GID

- Permessi speciali ed effetti speciali
- umask
- chown e chgrp
- 104.6 - Create and change hard and symbolic links [2]
 - symlink e hardlink in Linux
- 104.7 - Find system files and place files in the correct location [2]
 - type, which, whereis, FHS

101 - Architettura del sistema

101.1 - Determine and configure hardware settings [2]

Gli interrupts in Linux

Le periferiche comunicano con il processore tramite gli *interrupts* che 'interrompono' il processore per comunicare cosa stanno facendo (ad esempio l'interrupt della tastiera interrompe il processore ogni volta che si digita un tasto).

Il file `/proc/interrupts` (rigenerato ad ogni avvio del sistema) contiene (da sinistra a destra):

- il numero assegnato all'interrupt;
- quante volte questo ha interrotto il processore e quale processore/cpu;
- tipologia di interrupt;
- il nome dell'interrupt.

CPU0				
0:	2	IO-APIC	2-edge	timer
1:	9	IO-APIC	1-edge	i8042
6:	2	IO-APIC	6-edge	floppy
7:	0	IO-APIC	7-edge	parport0
8:	1	IO-APIC	8-edge	rtc0
9:	0	IO-APIC	9-fasteoi	acpi
12:	15	IO-APIC	12-edge	i8042
14:	0	IO-APIC	14-edge	ata_piix
15:	31143697	IO-APIC	15-edge	ata_piix
16:	143656652	IO-APIC	16-fasteoi	vmwgfx
17:	157153902	IO-APIC	17-fasteoi	ioc0
18:	172010897	IO-APIC	18-fasteoi	eth0
...				

ESERCIZIO

Individua l'interrupt della tastiera. Non è necessario consultare manuali online.

iports e DMA in Linux

File `/proc/ioports`

Ogni periferica per funzionare ha bisogno di un driver.

Quando una periferica viene rilevata, il sistema operativo carica il driver opportuno al suo funzionamento.

Il driver, assegna alla periferica una zona di memoria nello spazio indirizzabile dalla CPU.

Il file `/proc/ioports` elenca le zone di memoria che ciascun driver ha assegnato a ciascuna periferica.

Questi sono gli indirizzi su cui il processore può leggere e scrivere per interagire con le periferiche.

Ciascuno di questi indirizzi non può collidere o essere condiviso con altre periferiche o incorreremmo in un bellissimo "undefined behaviour".

File `/proc/dma`

Questo file indica i canali ISA DMA (Industry Standard Direct Memory Access) **attivi e collegati al BUS ISA**, come ad esempio controller IDE, Floppy, porte parallele e altre periferiche legacy. Sui sistemi moderni il BUS ISA non è più fisicamente presente ma è generalmente emulato dal southbridge.

Il concetto di **Direct Memory Access** consiste nel permettere ad una periferica di scambiare dati su un'area di memoria condivisa senza l'intermediazione della CPU, ovvero senza che sia la CPU a spostare i dati tra la periferica e la RAM, sprecando cicli inutilmente.

File `/proc/iomem`

Nei sistemi moderni, i mapping **DMA** delle periferiche sono consultabili all'interno del file `/proc/iomem`. Qui possiamo trovare le periferiche più disparate, dalle schede di rete alla schede audio e video. Per maggiori dettagli consiglio di leggere [la documentazione sul sito del kernel](#)

Dispositivi e drivers in Linux

Dispositivi ColdPlug e HotPlug

- Dispositivi **cold plug**: devono essere collegati e scollegati quando il computer è spento.
- Dispositivi **hot plug**: si possono collegare a computer acceso (ad esempio le periferiche usb).

Per elencare i dispositivi collegati sul bus PCI:

- `$ lspci`
- `$ lspci -tv #mostra le gerarchie ad albero`

Per elencare i dispositivi collegati sul bus USB:

- `$ lsusb` oppure
- `$ lsusb -tv` #per la rappresentazione ad albero

ESERCIZIO

Individua la tastiera sul bus USB. Quale driver è stato caricato per gestirla?

La cartella `/dev/`

Contiene diversi tipi di dispositivi (fisici e virtuali) rappresentati come file.

Dal kernel 2.6 in poi, `udev` crea o rimuove automaticamente i device sotto `/dev/` quando vengono collegati o scollegati.

Sui sistemi che usano `systemd`, il demone `udev` è stato inglobato dal servizio `systemd-udevd`.

ESERCIZIO

Individua il *device-file* della tastiera USB all'interno di `/dev/`.

Sapevi che è possibile **leggere e scrivere** su questo dispositivo?

Attivare, disattivare e visualizzare i driver attivi

Indipendentemente che siano *hot-plug* o *cold-plug*, tutti i dispositivi hanno bisogno di un driver per funzionare.

I driver attualmente in uso sono visualizzati con `lsmod`, che elenca:

- il nome del modulo;
- la sua dimensione;
- quante istanze del modulo sono in uso;
- elenco di moduli che ne fanno uso (può essere *incompleto*);

Il comando

- `modinfo <nome_modulo>`

Restituisce le generalità del modulo specificato, il suo percorso sul filesystem, il numero di versione e molto altro.

ESERCIZIO

Individua il modulo kernel con maggior numero di istanze attive.

ESERCIZIO (avanzato)

Stampa a schermo il modulo kernel con maggior numero di istanze attive.

I moduli che non sono utilizzati da nessun dispositivo possono essere rimossi dalla memoria tramite:

- # `rmmod <nome-modulo>` oppure
- # `modprobe -r <nome-modulo>`

In aggiunta, è possibile **inibire il caricamento di un modulo** in maniera persistente dal prossimo reboot inserendo una riga

- `blacklist <nome-modulo>`

All'interno del file `/etc/modprobe.d/blacklist.conf` (creandolo se necessario).

Tutti i moduli disponibili si trovano all'interno della directory `/lib/modules/<versione-kernel>/` e posso cercarli tramite il comando:

- # `find /lib/modules/$(uname -r)/ -iname "<nome-modulo>*.ko"`

Per inserire (attivare) un modulo invece posso usare `insmod` specificando il percorso completo:

- # `insmod <full-path-to-module>`

Se il modulo che voglio inserire è sotto `/lib/modules/<versione-kernel>/` posso evitare di specificare l'intero percorso usando `modprobe`:

- `modprobe <nome-modulo>`

ESERCIZIO

Individua un modulo kernel che possa essere rimosso in sicurezza, rimuovilo e re-inseriscilo.

Configurazione di un modulo

Un modulo kernel è essenzialmente un driver e ogni driver può avere dei parametri di configurazione specifici. Questi, se presenti, sono documentati nella documentazione ufficiale del modulo in questione.

La configurazione di un modulo può essere fatta con una riga in `/etc/modprobe.conf` o creando un apposito file di configurazione nella directory `/etc/modprobe.d/`.

Ad esempio `/etc/modprobe.d/droidcam.conf` che contiene:

```
options v4l2loopback_dc width=640 height=480
```

Configurerà il modulo `droidcam` per generare una fotocamera virtuale della risoluzione specificata.

Alternativamente, è possibile specificare queste **opzioni al caricamento del modulo** con la sintassi:

- `modprobe nome_modulo param1=val1`

Oppure **in fase di boot**, appendendole alle opzioni del kernel con questa sintassi:

- `linux ... nome_modulo.param1=val1`

Per **elencare tutti i parametri di configurazione supportati** da un modulo usiamo il comando

- `modinfo -p.`

I **valori attuali dei parametri** sono esposti come file di testo all'interno di:

- `/sys/module/<nome_modulo>/parameters/`

Questi pseudo-file possono essere sia leggibili che *scrivibili*. Scrivere un valore al loro interno equivale a cambiare in tempo reale il valore del parametro rappresentato dal file. Nota che non tutti i moduli supportano la modifica di qualunque parametro di configurazione a runtime.

ESERCIZIO

Quali parametri di configurazione supporta il modulo `usbhid`?

Quali valori sono attualmente impostati?

ESERCIZIO

Imposta il parametro `mousepoll` del modulo `usbhid` al valore '2' (equivalente a 500Hz ovvero 1000/2) e verifica che il cambiamento sia stato applicato al modulo caricato.

ESERCIZIO (avanzato)

Abilita il logging dei 'keypress' della tastiera, se il modulo in uso lo permette.

Indizio: è generalmente possibile in macchina virtuale.

La cartella `/sys/`

È il punto di montaggio predefinito per il filesystem virtuale `sysfs`, che contiene informazioni sui dispositivi, suddivisi per categoria:

```
ls /sys/
block  class  devices  fs      kernel  power
bus     dev    firmware  hypervisor  module
```

I moduli che ne fanno uso, possono esporre dettagli sullo stato interno delle periferiche e parametri di configurazione accessibili e modificabili dall'utente o da altri software.

È possibile **verificare** se `sysfs` è montato con il comando `mount | grep sys`.

In caso non lo fosse, è possibile **montarlo** manualmente tramite `mount -t sysfs sysfs /sys`.

Ad esempio, su un [Surface Pro 3](#), il driver dell'accelerometro espone le coordinate di rotazione tramite [sysfs](#) al percorso:

```
/sys/devices/pci0000:00/INT33C2:00/i2c-0/i2c-
MSHW0030:00/0018:045E:07C4.0002/HID-SENSOR-
200073.2.auto/iio:device5/in_accel_y_raw
```

Questo percorso è (sicuramente?) documentato nel modulo kernel interessato, ma è decisamente più immediato **cercare l'accelerometro** direttamente nel percorso [/sys/](#):

```
find /sys/ -iname "*accel*" 2>/dev/null
```

Analogamente, il controller della luminosità dello schermo avrà un percorso simile in [/sys/](#) ma in questo caso, oltre a **leggere** il valore attuale di luminosità, sarà anche possibile **scrivere** un valore arbitrario per impostare un livello di luminosità differente:

```
echo 2500 > /sys/class/backlight/intel_backlight/brightness
```

Il valore [2500](#) è il valore massimo supportato, riportato dal driver stesso in un file differente chiamato [max_brightness](#):

```
cat /sys/class/backlight/intel_backlight/max_brightness
2500
```

Dovrebbe essere ormai chiaro che i "file" in [/sys/](#) non sono veramente file ma interfacce per interagire con i moduli del kernel in modalità testuale.

D-Bus

Fornisce due canali di comunicazione per le applicazioni:

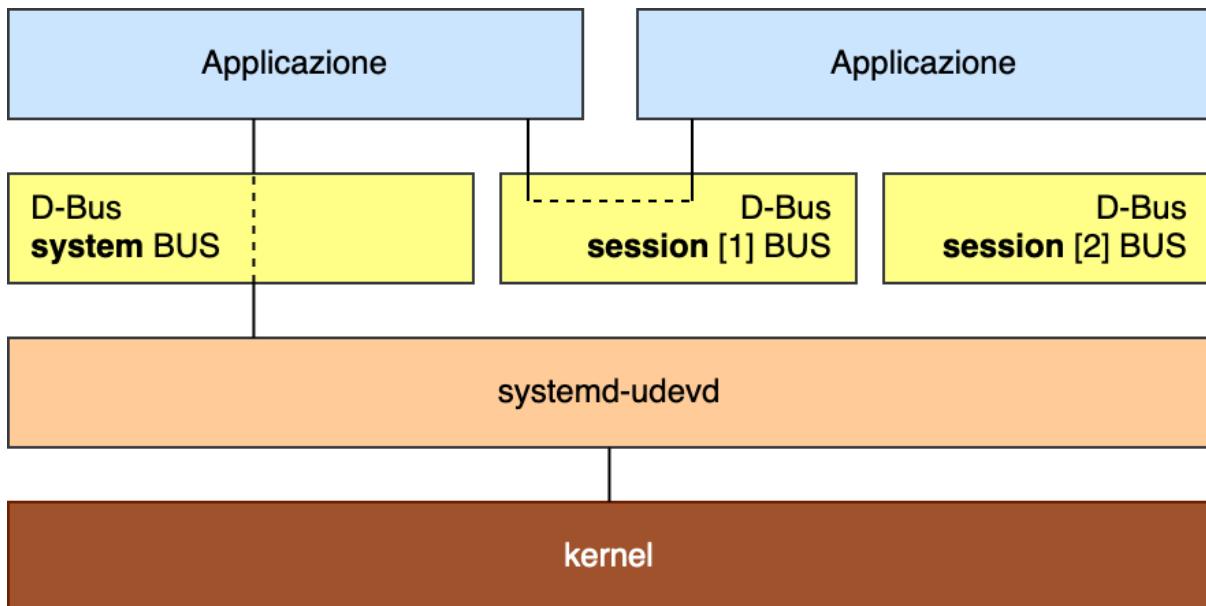
System Bus:

Permette alle applicazioni di **comunicare con l'hardware** tramite [HAL](#) (Harware Abstraction Layer, poi inglobato in [udev](#), a sua volta inglobato in [systemd-udevd](#)).

Lo scopo principale di questo bus è permettere alle applicazioni di essere notificate quando una nuova periferica viene collegata, scollegata o subisce un cambiamento di stato. GNOME ad esempio può utilizzarlo per aprire il file manager quando viene collegata una chiavetta USB.

Session Bus:

Permette alle applicazioni di **scambiarsi messaggi** nel contesto di una sessione di login. Possono esserci zero o più [session bus](#), uno per ciascun utente collegato.



Approfondimento: intercettare le comunicazioni su dbus

A questo scopo, possiamo utilizzare `busctl`.

- `busctl --user list`

Elenca tutti i servizi sul bus di sessione.

Se usiamo **Gnome Shell** come Desktop Environment, troveremo numerose voci con prefisso `org.gnome.*`.

Per monitorare un servizio specifico, utilizziamo l'opzione `monitor`:

- `busctl --user monitor org.gnome.Shell`

Ora il cursore è in attesa di comunicazioni sul bus. Se ad esempio, alziamo oabbassiamo il volume dal tasto fisico sulla tastiera, vengono generati due messaggi.

Il primo è un *segnalet* relativo alla pressione della scorciatoia da tastiera (`Member=AcceleratorActivated`) con tanto di riferimento al dispositivo di provenienza (`/dev/input/event20`):

```

▶ Type=signal Endian=l Flags=1 Version=1 Cookie=9148
Timestamp="Sat 2023-09-09 13:37:43.947526 UTC"
  Sender=:1.14 Destination=:1.57 Path=/org/gnome/Shell
  Interface=org.gnome.Shell Member=AcceleratorActivated
  UniqueName=:1.14
  MESSAGE "ua{sv}" {
    UINT32 211;
    ARRAY "{sv}" {
      ...
      DICT_ENTRY "sv" {
        STRING "device-node";
      }
    }
  }

```

```

        VARIANT "s" {
            STRING "/dev/input/event20";
        };
    };
};


```

Il secondo è la chiamata di un metodo (`method_call`) per mostrare il "popup" del volume all'utente (`Member=ShowOSD`), seguito dalle informazioni sul nuovo livello di volume: canale `Line Out` e livello 63% (`0, 630814`).

```

▶ Type=method_call Endian=l Flags=1 Version=1 Cookie=180
Timestamp="Sat 2023-09-09 13:37:43.947824 UTC"
Sender=:1.57 Destination=:1.14 Path=/org/gnome/Shell
Interface=org.gnome.Shell Member>ShowOSD
UniqueName=:1.57
MESSAGE "a{sv}" {
    ARRAY "{sv}" {
        DICT_ENTRY "sv" {
            STRING "icon";
            VARIANT "s" {
                STRING "audio-volume-medium-
symbolic";
            };
        };
        DICT_ENTRY "sv" {
            STRING "label";
            VARIANT "s" {
                STRING "Line Out";
            };
        };
        DICT_ENTRY "sv" {
            STRING "level";
            VARIANT "d" {
                DOUBLE 0,630814;
            };
        };
    };
};


```

Non sono richiesti particolari privilegi per leggere il bus di sessione, perciò *qualunque software nella stessa sessione utente può intercettare questi eventi e leggerne il contenuto.*

Alla stessa maniera, possiamo usare `busctl` o le API di qualunque linguaggio supportato per *emettere segnali o chiamare metodi* su qualunque servizio sul bus di sessione.

101.2 - Boot the system [3]

BIOS vs UEFI

- **BIOS**: Basic Input Output System - può indirizzare solo 16bit.
- **UEFI**: Unified Extensible Firmware Interface - più moderno e flessibile.

Sequenza di boot su sistemi BIOS

1. Viene eseguito il POST (Power On Self Test)
2. Viene caricato lo STAGE 1 boot loader dall'MBR (i primi 512 byte del disco)
3. Lo STAGE 1 bootloader avvia lo STAGE 2 boot loader
4. Lo STAGE 2 boot loader avvia il kernel, che può caricare un **initramfs**, se presente.
5. Il kernel avvia il **sistema di init**.

Tipicamente abbiamo:

POST --> GRUB STAGE 1 --> GRUB STAGE 2 --> LINUX + [initramfs] --> Systemd.

Sequenza di boot su sistemi UEFI

1. Viene eseguito il POST (Power On Self Test)
2. Vengono caricate le voci di avvio dall'**NVRAM** (Non-Volatile RAM)
3. Viene lanciato l'**esegubile EFI** indicato alla voce di avvio selezionata.
Questo deve risiedere su una **partizione FAT32** con il flag **esp** attivo.
4. L'eseguibile EFI è un bootloader e avvia il kernel, che può caricare un **initramfs**, se presente.
5. Il kernel avvia il **sistema di init**.

Tipicamente abbiamo:

POST --> NVRAM lookup --> GRUB.EFI --> LINUX + [initramfs] --> Systemd.

initrd e initramfs

Sono piccoli "dischetti virtuali" caricati in memoria all'avvio del kernel e contenenti un filesystem base di appoggio e un set minimale di moduli kernel aggiuntivi funzionali all'avvio del sistema.

- **initrd**: *initial ram disk* - usato fino al kernel 2.4
- **initramfs**: *initial ram filesystem* - in uso dal kernel 2.6 in poi.

Se un driver per qualche ragione deve essere disponibile già dalle prime fasi di boot, lo si include all'interno dell'**initramfs** in modo da essere caricato prima di tutto il resto.

Alcuni esempi di driver generalmente inclusi nell'**initramfs**:

- **/drivers/acpi/video.ko** : grafica integrata (per vedere qualcosa a schermo)
- **/drivers/hid/usbhid/usbmouse.ko** : mouse (USB)
- **/drivers/hid/usbhid/usbkbd.ko** : tastiera (USB)

- `/fs/btrfs/btrfs.ko` : filesystem in uso sulla partizione root (per completare l'avvio)
- ...

È possibile ispezionare il contenuto completo con '`lsinitrd`' e '`lsinitramfs`' rispettivamente.

Il file `initramfs` viene generato con `mkinitcpio` secondo le regole definite in `/etc/mkinitcpio.conf` oppure tramite `dracut` nelle distribuzioni Redhat-like e via dicendo.

Bootloaders

Principali bootloader per il sistema BIOS:

- **LiLo** (Linux Loader)
- **SysLinux** (Live CD boot ecc)
- **GRUB** (GNU GRand Unified Bootloader) ora chiamato **GRUB Legacy**

Elenco dei bootloader per UEFI:

- **GRUB2.0** (l'evoluzione di GRUB legacy, può fare il boot anche su BIOS)
- **systemd-boot** (il boot loader fornito dalla suite systemd)
- **ELILO** (EFI Lilo)

Chain loading

Quando deve lanciare `windows`, GRUB in realtà non fa altro che caricare il bootloader di `windows`, non essendo in grado di avviare propriamente `windows`.

La pratica di usare un boot loader per avviare un altro boot loader è detta **chain loading**, o caricamento a catena (ma non ditelo in italiano perché suona malissimo 😊)

Il sistema di init

Il primo processo ad essere lanciato dal kernel è il sistema di '`init`'. Un processo incaricato di **avviare tutti gli altri processi** per completare l'avvio del sistema. Essendo il primo processo ad essere lanciato, questo avrà sempre PID (Process ID) = 1.

In linux esistono più sistemi di init.

Lo storico standard 'de facto' è stato `System V` per lungo tempo.

System V divideva il processo di avvio in **runlevels**:

Esistevano 6 diversi runlevel:

- '`0`' : spegnimento.
- '`1`' : single-user (per manutenzione e recovery).
- '`2`' : multi-user (NO networking).
- '`3`' : multi-user (networking e sessione testuale).
- '`4`' : non in uso.

- '5' : multi-user completo (compreso la sessione grafica).
- '6' : reboot.

Per conoscere il `runlevel` attivo, possiamo usare il comando `runlevel`.

Per verificare qual'è il **runlevel predefinito** e cosa viene avviato in ciascun *runlevel*, consultiamo il file `/etc/inittab`.

Questo sistema non è più in uso da lungo tempo, anche se la maggior parte dei sistemi di init moderni continuano a riferirsi al concetto di runlevel per ragioni di retrocompatibilità.

Systemd è il nuovo standard 'de facto' attualmente in uso sulla maggior parte delle distribuzioni moderne e i runlevel 'classici' sono stati sostituiti dai cosiddetti `boot target`:

- `runlevel '0'` --> `poweroff.target`
- `runlevel '1'` --> `rescue.target`
- `runlevel '2'` --> `runlevel2.target`
- `runlevel '3'` --> `multi-user.target`
- `runlevel '4'` --> `runlevel4.target`
- `runlevel '5'` --> `graphical.target`
- `runlevel '6'` --> `reboot.target`
- `default.target` è un collegamento al `.target` attualmente impostato come predefinito.

Per verificare qual'è il **boot target predefinito** usiamo il comando `systemctl get-default`. Per impostare il **boot target predefinito** usiamo il comando `systemctl set-default <..>`.

I boot target sono più genericamente delle **systemd units**. Esistono diversi tipi di *systemd unit* a seconda dello scopo:

- `.target` : per avviare il sistema in maniera parziale o totale.
- `.service` : per definire l'avvio dei servizi e relativo ordine.
- `.timer` : per definire attività ricorrenti da eseguire con una periodicità predefinita.
- `.device.mount` : per definire regole di montaggio di dispositivi di archiviazione di massa.
- `.socket` : per definire l'avvio di un servizio in ascolto su un socket quando viene ricevuta la prima richiesta sul socket stesso.
- ...

Nei capitoli successivi vedremo le principali caratteristiche di `systemd` in maggior dettaglio.

Operazioni di recovery e diagnostica al boot

Quando il sistema non è in grado di avviarsi, è possibile tentare il recupero con le seguenti tecniche.

Avvio in modalità rescue

In caso di emergenza, è sempre possibile avviare il sistema su un runlevel differente modificando le impostazioni di lancio del kernel dal bootloader.

Sulla schermata di GRUB:

1. Spostarsi sulla voce di boot scelta
2. Premere 'e' per modificare le opzioni di lancio
3. Individuare la riga `linux...`
4. Aggiungere in fondo il numero corrispondente al runlevel desiderato (1,2,3,5).
5. Premere CTRL+X per avviare il sistema

Alternativamente, è possibile **interrompere il boot dopo il caricamento dell'initramfs** e ottenere una shell di root su questa base minimale:

Dalla schermata (edit) di GRUB,

1. Individuare la riga `linux...`
2. Aggiungere in fondo `rd.break`
3. Premere CTRL+X per avviare il sistema
4. Il sistema entra in "emergency mode" e otteniamo una shell di root sull'"initramfs".
5. Da qui è possibile **montare il filesystem in scrittura** con `mount -o remount,rw /sysroot`
6. Ora possiamo spostarci sulla root del filesystem completo: `chroot /sysroot`
7. Da qui è possibile svolgere qualunque operazione come utente root.

Reset della password

In caso di smarrimento della password, è possibile ottenere l'accesso completo (`root`) al sistema modificando le opzioni di avvio del kernel con la stessa procedura del punto precedente.

Se, alla fine della linea del kernel di GRUB, aggiungiamo '`init=/bin/sh`', il kernel lancerà, con il massimo livello di privilegio, l'eseguibile della shell `sh` anziché il sistema di init, bypassando di fatto il meccanismo di autenticazione!

A questo punto è possibile impostare / reimpostare la password di qualunque utente con il comando:

```
passwd <utente>
```

ESERCIZIO

Esegui il reset della password di root.

Consultare opzioni di avvio e configurazione del kernel

Il file `/proc/cmdline` contiene le opzioni di avvio del kernel per il boot corrente. Le modifiche "temporanee" illustrate nei due punti precedenti **sono verificabili in questo modo**.

Sui kernel che lo supportano, `zcat /proc/config.gz` mostra la configurazione a tempo di compilazione del kernel attualmente in esecuzione.

Diagnosi e troubleshooting

Tutti gli eventi in linux sono raccolti all'interno di diversi file di *log*.

- il comando `dmesg` mostra i messaggi del kernel. Questi sono memorizzati in un "ringbuffer" in RAM e rimangono accessibili *fino al riavvio successivo o finché non viene eseguito `dmesg --clear`*.
- il comando `journalctl -b` mostra tutti i messaggi dall'ultimo boot (sui sistemi con `systemd`)
- il file `/var/log/boot.log` sulle distro che ne fanno uso (ES: Fedora), contiene tutti i messaggi di boot (comunque reperibili anche da `journalctl -b SYSLOG_PID=1`)
- il file `/var/log/messages` è un log globale da cui partire per cercare eventuali errori (sono presenti eventi dbus, eventi kernel ed altro). Alcune distribuzioni (ES: Fedora) non fanno uso di questo file di log, in favore di `systemd`.

Leggere i log con `journalctl`

'`journalctl`' è l'aggregatore dei log di sistema fornito da '`systemd`'.

Non è pratico usare `journalctl` senza opzioni in quanto il suo output risulta essere un'infinito elenco di log provenienti da qualsiasi elemento del sistema pertanto analizziamo le opzioni principali, utili a selezionarne gli elementi da visualizzare.

- `$ journalctl -b # mostra i log dal boot`
- `$ journalctl -br # mostra i log dal boot in ordine inverso (opzione '-r')`
- `$ journalctl --list-boots # mostra tutti i boot della macchina indicando un numero identificativo all'inizio della riga`
- `$ journalctl -b <numero-boot> /# per visualizzare di un determinato boot`
- `$ journalctl --since="1 hour ago" # mostra i log dell'ultima ora`
- `$ journalctl --since="20 minutes ago" # mostra i log degli ultimi 20 minuti`
- `$ journalctl --since="2022-10-31" # mostra i log della data indicata`
- `$ journalctl --since="1 hour ago" --until="20 minutes ago" # mostra i log dell'ultima ora fino a 20 minuti fa`
- `$ journalctl _UID=<user-id> # mostra i log di un determinato utente`

NOTE:

Le righe dell'output di '`journalctl`' non vanno a capo ma possono essere visualizzate se troppo lunghe utilizzando i tasti cursore '`<RIGHT>`' e '`<LEFT>`'.

L'opzione '`-k`' equivale all'opzione '`--dmesg`' e mostra (appunto) i messaggi del kernel (equivale al comando '`dmesg`').

L'opzione '`-u <nome-unita>`' (oppure '`--unit <nome-unita>`') visualizza i log di quella particolare unità (molto usato per mostrare i log di servizi specifici).

Il comando '`systemctl`' lanciato da solo visualizza le unità in esecuzione.

L'output di '`journalctl`' è ricercabile scrivendo '`/parola chiave`' + '`<ENTER>`' (e poi '`N`' per cercare la ricorrenza successiva, '`n`' per cercare a ritroso).

Possiamo anche **leggere i log in tempo reale** aggiungendo l'opzione `-f` al comando `journalctl`.

ESERCIZIO

Con l'aiuto del `man journalctl`, individua l'opzione per elencare i log dei boot precedenti e stampa il suddetto elenco. Quanti elementi compaiono e Perché?

101.3 - Change runlevels / boot targets and shutdown or reboot system [3]

Cambiare runlevel all'avvio

Come accennato in precedenza, è possibile cambiare runlevel al boot inserendo il numero di runlevel desiderato a fine riga nelle opzioni del kernel in GRUB.

Cambiare runlevel a sistema avviato

Con `systemV` era possibile passare al runlevel desiderato (ad es. 3) con il comando `init 3` o `telinit 3`.

Per sapere in quale runlevel siamo, digitiamo il comando `runlevel` o `who -r`.

Su `systemd` il comando analogo diventa `systemctl isolate multi-user.target`.

Il comando `init <parametro>` funziona ancora per ragioni di retrocompatibilità, ma l'implementazione sottostante è un semplice collegamento a `systemd`, così come l'eseguibile `/sbin/init` lanciato dal kernel come primo processo (PID 1):

```
[morro@master ~]$ ls -lah /sbin/init
lrwxrwxrwx 1 root root 22 27 set 15.11 /sbin/init ->
./lib/systemd/systemd
```

Cambiare il runlevel predefinito

Sui sistemi che utilizzano `systemd` è possibile cambiare il runlevel predefinito tramite:

- `# systemctl set-default <nuovo-runlevel-predefinito>`

Mentre per i sistemi che utilizzano `System V init` è necessario modificare il numero di runlevel di default all'interno del file `/etc/inittab` e precisamente nella seguente riga:

- `id:5:initdefault:`

ESERCIZIO [1/2]

Imposta il boot target predefinito a `multi-user.target` e passa immediatamente al

medesimo runlevel senza riavviare.

ESERCIZIO [2/2]

Riavvia il sistema forzando il caricamento di `graphical.target` (o equivalente) a discapito del runlevel impostato come predefinito al punto precedente.

File `/etc/inittab` (systemV)

Oltre al runlevel di default, all'interno del file `/etc/inittab` possono essere configurati, per ogni runlevel, gli script di avvio che vengono lanciati con la seguente sintassi:

```
id:runlevel:mode:command
```

Ad esempio, per avviare il gestore di login `xdm` al runlevel 5:

```
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Maggiori informazioni sui parametri utilizzati (`respawn`, `wait`, `once`, ...) sulla pagina di manuale di `inittab`.

NOTA

il file `/etc/inittab` non è più in uso sulle distro che usano `systemd`.

La cartella `/etc/init.d/` (systemV)

La cartella `/etc/init.d/` contiene tutti gli script che possono essere lanciati con il passaggio ad un determinato runlevel.

Le cartelle `/etc/rc<numero-runlevel>.d/` invece contengono i collegamenti simbolici agli script che si trovano in `/etc/init.d/` che verranno lanciati (se lo script inizia con '`S`') oppure "uccisi" (lo script inizia con '`K`').

'`sysv-rc-conf`' è lo strumento che permette di configurare quali servizi vengono lanciati per quali runlevel, è la stessa cosa che creare/eliminare manualmente i collegamenti di cui sopra.

Comandi shutdown e wall

Il comando `wall "messaggio"` manda un messaggio in broadcast a tutti gli utenti su tutte le TTY.

Per spegnere la macchina è preferibile usare il comando `shutdown` che:

1. Avvisa tutti gli utenti collegati tramite `wall`;
2. Cambia il runlevel tramite `init`;

3. Quest'ultimo comando manda un messaggio di **SIGTERM** a tutti i processi che hanno 5 secondi per terminare correttamente;
4. Se i processi non terminano, manda un segnale di **SIGKILL** che li uccide in maniera brutale.

Un paio di esempi:

- `# shutdown 17:01 "messaggio" # programma uno shutdown che verrà anticipato da un messaggio personalizzato`
- `# shutdown -r # programma un reboot che verrà anticipato dal messaggio predefinito`
- `# shutdown -c # annulla uno shutdown o reboot programmato`

Come funziona systemd

L'init system **systemd** punta ad avviare ciascun servizio solo e soltanto quando questo viene effettivamente richiesto, e non a priori. Tutto questo rende, almeno teoricamente, l'avvio del sistema più rapido.

Systemd configura **servizi**, **risorse** e relative **dipendenze** tramite file chiamati **unit**.

Ogni **unit** di systemd ha un'estensione che in qualche modo la identifica: `'.service'`, `'.socket'`, `'.device'`, `'.mount'`, `'.automount'`, `'.swap'`, `'.target'`, `'.path'`, `'.timer'`, `'.snapshot'`, `'.slice'`, `'.scope'` e altre ancora

Il comando principale per controllare lo stato delle **unit** è **systemctl**:

- `# systemctl status <nome-unit>`
- `# systemctl stop <nome-unit>`
- `# systemctl start <nome-unit>`
- `# systemctl enable <nome-unit>`
- `# systemctl disable <nome-unit>`

Ad esempio, per controllare lo stato del servizio **ssh**:

```
systemctl status ssh.service
```

Restituisce:

```
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled;
   vendor preset: enabled)
     Active: active (running) since Sun 2023-08-20 21:39:52 CEST;
           21h ago
       Docs: man:sshd(8)
              man:sshd_config(5)
     Process: 2334 ExecStartPre=/usr/sbin/sshd -t (code=exited,
           status=0/SUCCESS)
    Main PID: 2372 (sshd)
      Tasks: 1 (limit: 9274)
```

```

Memory: 3.7M
CPU: 561ms
CGroup: /system.slice/sshd.service
└─2372 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100
startups

Aug 20 21:39:52 pve sshd[2372]: Server listening on :: port 22.
Aug 20 21:39:52 pve systemd[1]: Started OpenBSD Secure Shell
server.
Aug 21 09:53:04 pve sshd[538472]: Accepted password for root from
192.168.1.8 port 62316 ssh2
Aug 21 09:53:04 pve sshd[538472]: pam_unix(sshd:session): session
opened for user root(uid=0) by (uid=0)

```

PRO TIP

In caso di boot lento, possiamo analizzare quanto ogni servizio ha impiegato ad avviarsi con il comando `$ systemctl-analyze blame`

ESERCIZIO

Stai scrivendo uno script e devi controllare se un determinato servizio è attivo.

`systemctl status nome_servizio` si è rivelato inadatto a questo scopo in quanto eccessivamente "verboso" nel suo output. Cosa puoi usare al suo posto?

Systemd e Systemd User

Systemd prevede due modalità di operazione:

- A livello di sistema (impostazione predefinita)
- A livello di utente (specificando `--user`)

Tutte le operazioni svolte con `systemctl` sono intese a livello di **sistema** e richiedono **privilegi elevati** qualora si volesse apportare modifiche.

Un *utente non privilegiato* può comunque creare le proprie systemd unit e verificare lo stato di tutte le unità con l'opzione `--user`:

- `systemctl --user list-units --type=service`

Ad esempio, mostra tutti i servizi attivi che eseguono con i normali privilegi dell'utente.

Dove si trovano le systemd unit

I file di unit di `systemd` sono installati in:

- `/usr/lib/systemd/system` (per le unit di sistema) e
- `/usr/lib/systemd/user` (per le unit a privilegi ridotti)

NOTA

Tutte le unit sotto la gerarchia `/usr/lib/` sono installate dal package manager e **non**

devono essere modificate direttamente in quanto possono essere **sovraffritte** al successivo aggiornamento del pacchetto che le ha installate in primo luogo.

Le unit create sotto la gerarchia `/etc` **Sono invece modificabili**:

- `/etc/systemd/system/` (unit di sistema) e
- `/etc/systemd/user/` (unit utente)

È anche possibile creare file drop-in all'interno di:

- `/etc/systemd/system/nome_unit.d/override.conf` e
- `/etc/systemd/user/nome_unit.d/override.conf`

Per **sovrascrivere singole voci di configurazione**, mantenendo la unit "originale" come template per tutto il resto.

PRO TIP

Per *modificare* una unit in modalità "drop-in" possiamo usare `systemctl edit nome_unit`.

NOTA

Nessuno di questi percorsi è scrivibile da un utente non privilegiato!

Un utente non privilegiato può creare le proprie unit all'interno di:

- `~/.config/systemd/user/`

Creando il percorso se necessario.

Com'è fatta una systemd unit

Prendiamo in esempio il servizio `ssh` di cui prima. Il percorso della unit file è `/lib/systemd/system/ssh.service`.

Vediamo com'è scritta questa unit:

```
# cat /lib/systemd/system/ssh.service

[Unit]
After=network.target auditd.service    <-- Deve essere lanciato dopo
di loro

[Service]
EnvironmentFile=-/etc/default/ssh      <-- Carica le variabili da
qui
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS <-- COMANDO DA LANCIARE
CON OPZIONI
Restart=on-failure                      <-- Riavvia il servizio se
fallisce
Type=notify                                <-- Tipo di servizio
```

```
...  
[Install]  
WantedBy=multi-user.target <-- Avvia quando raggiungiamo multi-  
user.target
```

Consultare la [documentazione ufficiale](#) per maggiori dettagli sulle opzioni.

NOTA: dopo la creazione o modifica di una unit è necessario *ricaricare systemd* per renderlo al corrente della modifica, tramite il comando:

- `# systemctl daemon-reload` per i servizi di sistema, oppure
- `$ systemctl --user daemon-reload` per i servizi utente.

ESERCIZIO [1/2]

Crea una systemd "service unit" che effettua il "ping" di "morrolinux.it", avvia il servizio, controllane lo stato e segui il log.

ESERCIZIO [2/2]

Crea una unit `--user` per un servizio che effettua il "ping" di "corsolinux.com", avvia il servizio (e abilita l'avvio automatico) e controllane stato e log come nel punto precedente.

Il demone acpid

E' un demone per gestire gli eventi ACPI (Advanced Configuration Power Interface).

Gestisce eventi come l'apertura/**chiusura del coperchio** del portatile, tasti speciali, connessione o **disconnessione dalla rete elettrica**, cambio di stato della carica della batteria e altro.

Da tener presente che certi *Desktop Environment* (come Gnome o KDE), oppure il *login manager* di `systemd` (`systemd-logind`) hanno i loro demoni e regole speciali per gestire questi eventi e che, questo demone potrebbe quindi andare in conflitto.

E' quindi probabile che questo demone sia disabilitato. Per controllare :

- `$ systemctl status acpid.service`

Le regole di `acpid` possono essere configurate con appositi file di configurazione all'interno della cartella `/etc/acpi/`.

Posto che `acpid` sia installato in primo luogo.

102 - Installazione di Linux e Package Management

102.1 - Design hard disk layout [2]

Alterare la tabella delle partizioni

I due principali strumenti per visualizzare e modificare lo schema delle partizioni sono **fdisk** e **parted**. Entrambi supportano sia il vecchio sistema **MBR** (Master Boot Record) che il più recente **GPT** (Guid Partition Table). Esiste anche un tool grafico per la modifica dello schema delle partizioni chiamato **gparted** che automatizza operazioni elementari come la creazione di una nuova partizione e di un filesystem su di essa.

Sia **fdisk** che **parted** supportano sia l'uso in modalità *non interattiva* che in modalità *interattiva*.

Alcuni esempi (modalità non interattiva):

- `# parted -l` # elenca i dischi presenti e relative partizioni
- `# parted /dev/sda print` # visualizza le partizioni del disco selezionato
- `# parted /dev/sda print free` # visualizza anche gli spazi vuoti
- `# parted /dev/sdb mklabel msdos` # elimina il contenuto di /dev/sdb e crea una tabella delle partizioni MBR
- `# parted /dev/sdb -a cylinder mkpart primary linux-swap 2 102` # crea una partizione tipo swap di 100MB allineata alla dimensione dei cilindri del disco a partire da 2MB
- `# mkswap /dev/sdb1` # crea il filesystem di swap sulla partizione appena creata
- `# parted /dev/sdb mklabel gpt` # cancella tutto il contenuto del disco e cambia la tabella delle partizioni in GPT

Utilizzo di Parted in modalità interattiva:

- `# parted` # senza digitare altro si entra in modalità interattiva
- `(parted) select /dev/sda` # seleziona il disco
- `(parted) print`
- `(parted) print free` # come prima ma si mettono solo i comandi
- `(parted) mkpart`
- `Tipo di partizione? [primaria/estesa]`
- `Tipo di file system? [ext2/linux-swap/...]`
- `Inizio? 2MB`
- `Fine? 102MB`
- `(parted) print`
- `(parted) q` # per uscire

NOTA

Creare una partizione di un certo tipo (ext2,ext4,linux-swap,...) in **fdisk** o **gparted** **riserva soltanto lo spazio** per quella partizione e **applica un'etichetta** per indicare il tipo di filesystem a cui è destinata la partizione. **Questa operazione non crea alcun filesystem** sullo spazio appena riservato.

Per creare il filesystem sulla partizione appena creata, usiamo l'utility **mkfs**.

Ad esempio:

- `# mkfs.ext4 /dev/sdb1` # crea un filesystem di tipo ext4 sulla partizione 1 del disco /dev/sdb

- # `mkfs -t ext4 /dev/sdb1` # stessa cosa, con una sintassi diversa.
- # `mkfs /dev/sdb1` # crea un filesystem di tipo **ext2 (default)** sulla partizione **1** del disco `/dev/sdb`

Ora è possibile montare la partizione con il comando `mount` e iniziare ad utilizzarla. Vedremo successivamente in dettaglio l'utilizzo del comando `mount`:

Per creare un filesystem di swap utiliziamo invece il comando `mkswap`:

- # `mkswap /dev/sdb1` # crea il filesystem di swap sulla partizione specificata

Consulta `man mkfs` e `man mkswap` per tutte le opzioni disponibili.

NOTA

Per attivare la partizione di swap ed iniziare ad utilizzarla è necessario usare il comando `swapon` che vedremo in dettaglio nei paragrafi successivi.

ESERCIZIO

Aggiungi un disco vergine al sistema (8G), crea una tabella GPT e una unica partizione con filesystem ext4 fino al 100% del disco. Assicurati di aver impostato correttamente anche la label della partizione!

ESERCIZIO (avanzato)

Monta il filesystem appena creato e crea al suo interno un file chiamato "morrolinux.it", poi smonta il filesystem. (richiede spirito d'iniziativa 😊)

Ridurre o espandere una partizione

Strada facendo, potrebbe rendersi necessario ridurre una partizione per permettere la creazione di una nuova partizione o estenderne una già esistente. **parted** e **fdisk** sono entrambi in grado di ridurre o espandere una partizione indicata a seconda dello spazio disponibile (in modalità interattiva è più semplice).

Dopo aver espanso una partizione con `fdisk` o `parted` non dimentichiamo di espandere anche il *filesystem* su di essa con il comando:

- # `resize2fs /dev/sdXY` # dove X e Y sono la lettera del device e la partizione scelta.

Alla stessa maniera, **prima di ridurre una partizione** riduciamo il suo *filesystem* con il comando:

- # `resize2fs /dev/sdXY <dimensione finale>`

La dimensione va espressa con l'unità di misura **G, M, K, ..**

IMPORTANTE

Può capitare che la tabella delle partizioni caricata in memoria **differisca da quella effettivamente scritta sul disco** se questa è appena stata riscritta. In questo caso dobbiamo riallineare le tabelle con il comando `partprobe /dev/sdXY`.

ESERCIZIO

Riduci la partizione creata in precedenza del 50% lasciando libero il restante spazio.

Assicurati di non corrompere il filesystem ext4 ridimensionando la partizione!

Swap

È un'area di memoria (su disco) dove i dati vengono 'travasati' quando finisce la RAM.

Per elencare tutte le partizioni e file di swap **attualmente in uso**, usiamo:

```
# swapon -s
```

Che elenca ciascun file o partizione di swap in uso con relativa dimensione, utilizzo attuale e priorità di attivazione. Un valore di *Priority* più alto indica una *maggior priorità*.

Filename	Type	Size	Used	Priority
/dev/dm-0	partition	974844	491268	-2
/swapfile	file	914844	0	-3

Le stesse informazioni sono consultabili dal file </proc/swaps>.

Swappiness

Serve ad impostare una soglia per quando il sistema inizia ad usare la SWAP. A seconda dei casi può far comodo che la swap venga attivata prima che la RAM si saturi completamente. Il valore attuale è consultabile nel file:

- \$ cat /proc/sys/vm/swappiness

Che restituisce un valore da 0 a 100, **più il valore è alto, più lo swapping sarà aggressivo**.

Impostando la swappiness a 0, la swap entrerà in funzione solo quando la RAM sarà effettivamente esaurita.

Per cambiare questo valore, è sufficiente sovrascrivere il contenuto del file **swappiness**:

- # echo 30 > /proc/sys/vm/swappiness/

Questa modifica avviene immediatamente, ma il valore ritorna allo stato originale al riavvio successivo!

Per rendere permanente il nuovo valore di swappiness, dobbiamo modificare </etc/sysctl.conf> come segue:

- vm.swappiness=80

Creare un'area di swap

La SWAP può risiedere su una partizione dedicata o in un file opportunamente creato.

Come regola generale per decidere quanta swap utilizzare, possiamo fare da 1 a 2 volte la dimensione della RAM, anche se, mano mano che cresce la RAM installata possiamo anche diminuire la dimensione della partizione swap. Se abbiamo in programma di utilizzare la *sospensione in RAM* serve un quantitativo di swap pari o superiore alla dimensione della RAM.

Per **creare un'area di SWAP** è sufficiente:

1. Creare una partizione o un file della dimensione desiderata
 - # `parted mkpart primary swap linux-swap 2M 102M` per creare una **partizione**, oppure
 - # `dd if=/dev/zero of=/swapfile bs=1M count=100 && chmod 600 /swapfile` per creare un **file**.
2. Creare il *filesystem di swap* con il comando `mkswap /file/o/partizione`
3. Ora è possibile **attivare l'area di swap** appena creata con il comando `swapon /file/o/partizione`.

Per **disattivare l'area di swap** usiamo analogamente il comando `swapoff /file/o/partizione`

Per **rendere permanente** l'attivazione della swap, possiamo inserire una nuova riga nel file `/etc/fstab`:

UUID=ID-DELLA-PARTIZIONE-DA-BLKID	swap	swap	defaults 0 0
-----------------------------------	------	------	--------------

oppure

/swapfile	swap	swap	defaults 0 0
-----------	------	------	--------------

Affronteremo `fstab` più in dettaglio nei capitoli successivi.

NOTA

nel file `fstab` è bene indicare dischi e partizioni con il relativo ID univoco ottenibile tramite il comando `blkid /dev/sdXY`.

PRO TIP

È possibile montare subito tutte le aree di swap configurate in `fstab` con un unico comando: `swapon -a`.

Analogamente, è possibile montare subito tutti i filesystem configurati in `fstab` con `mount -a`.

Per **cambiare la priorità di un'area di swap** usiamo:

- # `swapon /dev/sdb1 -p <priorita>`

Questo è immediato e vale solo fino al riavvio della macchina.

Per rendere la modifica persistente dobbiamo modificare il file `/etc/fstab` inserendo l'opzione `pri=<priorita>`

ESERCIZIO

Crea una partizione di SWAP nello spazio liberato in precedenza sul disco secondario, attiva la partizione con priorità 1 e fai in modo che sia persistente al prossimo reboot.

Progettare il partizionamento dei dischi

In Linux esiste uno standard di come deve essere strutturato il filesystem partendo dalla radice `(/)` chiamato **FHS** (Filesystem Hierarchy Standard). Non tutte le distribuzioni lo seguono alla lettera e in ogni dettaglio, ma la struttura generale sarà piuttosto chiara.

A seconda dei casi, può essere vantaggioso montare *determinati percorsi su partizioni separate*. Ad esempio:

- `/home/` per reinstallare il sistema senza eliminare i dati degli utenti e senza doverli migrare;
- `/var/` su un disco meccanico (non SSD) ad esempio, per ridurre l'usura dovuta alle continue scritture a cui è soggetta;
- `/boot/` per semplificare diagnostica e recovery se ad esempio il filesystem `/` è un modulo esterno al kernel (Btrfs, ZFS, ...)

Ci sono anche ragioni legate allo spazio disponibile:

La cartella `/var/log/` (ad esempio) può diventare molto grande quando qualche applicazione inizia a inviare messaggi log con una frequenza intensa per qualche errore. Questo può arrivare a saturare il disco in alcuni casi, creando complicazioni per tutti gli utenti collegati.

Teniamo a mente che è sempre possibile **spostare qualunque percorso su una nuova partizione**.

A grandi linee:

1. Creazione di una nuova partizione (e costruzione del filesystem)
2. Montaggio su un percorso temporaneo (vedremo in seguito come funziona)
3. Spostamento di tutti i file sulla nuova partizione (vedremo in seguito)
4. Inserimenti di una regola in `fstab` per il montaggio della nuova partizione sul vecchio percorso

A tal proposito,

Per elencare tutte le partizioni montate e relativo utilizzo, usiamo il comando `df`:

- `$ df -hT`

Per pesare il contenuto di una cartella, possiamo usare il comando '`du`' :

- `$ du -sh <cartella>/ # conteggio totale (dimensione cartella)`

- `$ du -sh <cartella>/*` # conteggio individuale (dimensione di ogni file e sottocartella)

Per **copiare tutto il contenuto di una cartella**, preservando permessi e attributi, usiamo:

- `cp -raf /percorso/di/origine /percorso/di/destinazione`

Alternativamente, è possibile **spostare la cartella** con il comando:

- `mv /percorso/di/origine /percorso/di/destinazione`

Nota che in questo caso non sono presenti opzioni per preservare gli attributi. **Come mai?**

ESERCIZIO (avanzato)

1. Crea uno snapshot della macchina virtuale\
2. Sposta il contenuto della cartella `/home` sulla partizione EXT4 creata in precedenza sul disco secondario\
3. Crea una regola in `fstab` per montare la suddetta partizione in `/home\`
4. Controlla che tutto sia corretto e riavvia.

Ha funzionato, o hai rotto tutto?

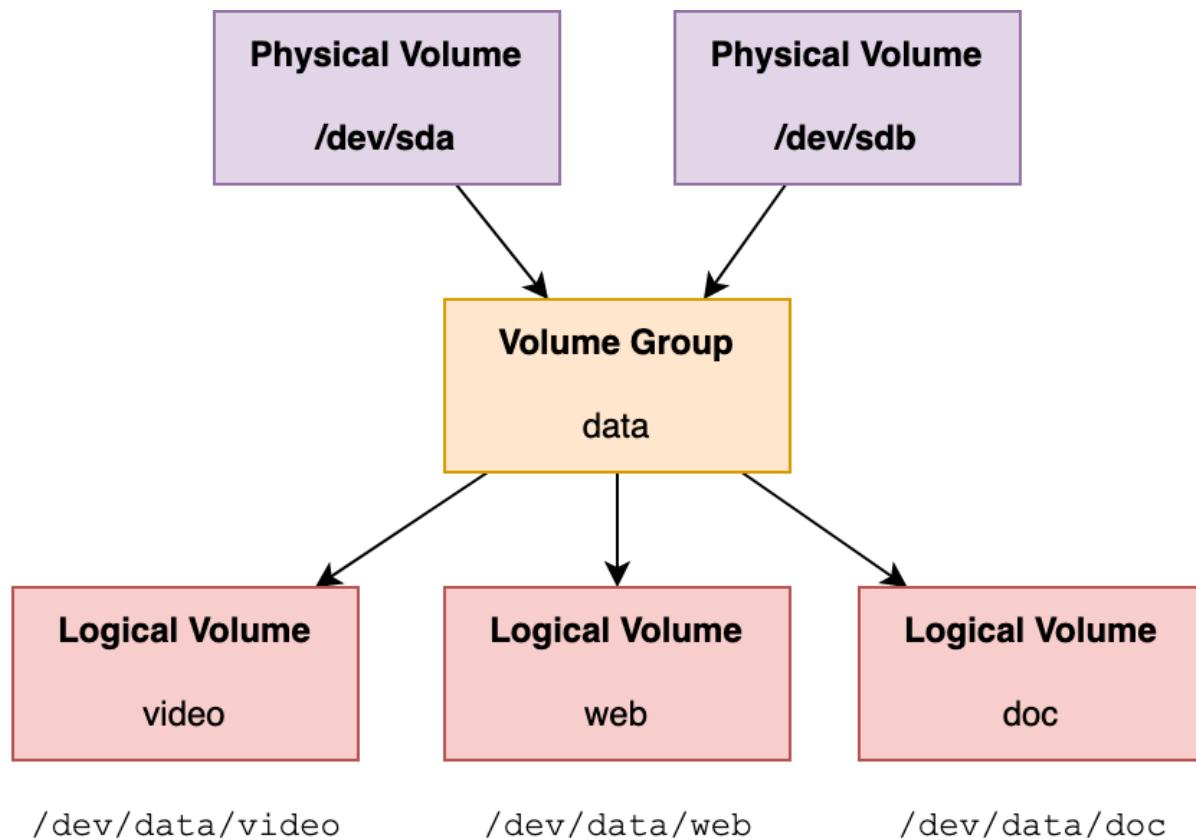
Cos'è e come funziona LVM (awareness)

LVM (Logical Volume Manager) permette di creare un livello di astrazione al di sopra del "dispositivo fisico" sottostante. Questo permette una maggiore flessibilità nella gestione dello storage senza doversi preoccupare dei dettagli dell'hardware sottostante.

Ad esempio, è possibile **espandere indefinitamente una partizione oltre alle dimensioni del disco** continuando ad aggiungere nuovi dischi, proteggere i dati da eventuali guasti tramite la **ridondanza**, *creare snapshot* per ricondursi ad uno stato precedente o **migrare tutti i dati** su un nuovo pool di dischi senza dover copiare alcun file e senza interruzioni di servizio.

A questo scopo, LVM crea le seguenti astrazioni:

- **Physical Volume (PV)** : è un disco "fisico" (es: `/dev/sdb`)
- **Volume Group (VG)** : è come un mega disco "virtuale" che aggrega tutti i PV (es: `/dev/data`)
- **Logical Volume (LV)** : è come una partizione sul VG (es: `/dev/data/video`)



Grazie a questa astrazione, è sempre possibile aggiungere nuovi *PV* per aumentare la dimensione del *VG* e conseguentemente espandere le ~~partizioni~~ pardòn, i *Logical Volume* indefinitamente.

NOTA

Con questo meccanismo, il filesystem non viene costruito su un singolo disco fisico, ma può risiedere su più dischi simultaneamente.

LVM: aggiungere e rimuovere dischi

Esistono 3 gruppi di comandi che si riferiscono ed agiscono rispettivamente su:

- (p)hysical (v)olumes, con i comandi che iniziano per '`pv...`'
- (v)olume (g)roup, con comandi che iniziano per '`vg...`'
- (l)ogical (v)olume, con i relativi comandi che iniziano per '`lv...`'

Ad esempio, per analizzare la situazione digitiamo:

- `# pvscan` # per elencare tutti i dischi fisici disponibili
- `# vgscan` # per elencare tutti i volume group presenti
- `# lvscan` # per elencare tutti i logical volume creati

Per iniziare ad usare LVM, sarà necessario:

1. Creare uno o più Physical Volume
2. Creare un Volume Group sui PV scelti

3. Creare uno o più Logical Volume sul VG creato

NOTA: Creando un Physical Volume, è consigliabile utilizzare *partizioni e non interi dischi*.

In breve:

1. Creo una nuova partizione (ad es: `/dev/sdb1`) etichettata *Linux LVM* (codice `8e`)
2. Creo il PV: `# pvcreate /dev/sdb1` (eventualmente specificando più dischi)
3. Creo il VG: `# vgcreate data /dev/sdb1` (creo un VG chiamato "data" sui PV creati)
4. Creo il LV: `# lvcreate -n video -L 1G data` (Creo un LV di 1GB chiamato "video" sul VG "data")

Per estendere un Logical Volume esistente:

1. Creo un nuovo PV come in precedenza (ad esempio su `/dev/sdc1`)
2. Estendo il Volume Group esistente: `# vgextend data /dev/sdc1`
3. Estendo il Logical Volume desiderato: `# lvextend /dev/data/video -L +1G`

Per ridurre un Logical Volume esistente: Prima di tutto dobbiamo fare in modo che la somma totale dei *logical volume* lasci spazio almeno per togliere il *physical volume*. Ad esempio, se ho 2 dischi da 1GB ciascuno e voglio scollarne uno, la somma di tutte le partizioni create (i *logical volume*) deve essere al massimo 1GB.

A grandi linee:

1. Mi assicuro di non avere più dati di quanti possa contenere senza il disco che sto per rimuovere
2. Riduco il filesystem sul LV: `# resize2fs /dev/data/video 1G` (siamo passati da 2G a 1G)
3. Riduco il logical volume: `# lvresize /dev/data/video -L -1GB`
4. Sposto eventuali dati dal PV che sto per rimuovere: `# pmove /dev/sdb1`
5. Riduco il VG: `# vgreduce data /dev/sdb1`
6. Rimuovo il PV: `# pvremove /dev/sdb1`

NOTA

LVM può essere usato in maniera interattiva digitando:

- `# lvm`
- `lvm> help`

ESERCIZIO (avanzato)

1. Ripristina la macchina virtuale allo snapshot precedente o ripristina il contenuto di `/home` sul disco principale.
2. Assicurati di rimuovere ogni riferimento in `fstab` alle partizioni del disco secondario.
3. Formatta il disco secondario in modo che abbia una unica partizione.
4. Trasforma la partizione appena creata in un physical volume LVM.

5. Crea sul physical volume un volume group chiamato "vg1" e due logical volumes: "home" e "var" con filesystem EXT4.
6. Copia o sposta l'attuale contenuto di `/home` e `/var` nei rispettivi volumi LVM, montando questi volumi all'avvio, similmente all'esercizio precedente.

La partizione EFI (ESP)

La prima cosa che fa un computer UEFI quando si accende è caricare le voci di boot dall'NVRAM. Queste non sono altro che collegamenti ad eseguibili EFI che si trovano su una *partizione EFI*. La partizione EFI è una comuniSSima partizione **FAT32** con il flag **esp** attivo.

PRO TIP

È possibile elencare, modificare e rimuovere le voci nell'NVRAM con **efibootmgr**.

Gli eseguibili EFI (LiLo, systemd-boot, GRUB, Winload, ...) sono bootloader "first stage" con il compito di lanciare il kernel e quindi il sistema operativo.

In GNU/Linux, la partizione UEFI è solitamente montata sotto `/boot/` o `/boot/efi/` ed entrando ci possiamo trovare altre cartelle, una per ogni boot loader installato, oltre alla cartella `BOOT/` dove risiede l'eseguibile predefinito di UEFI che, dopo aver installato Linux è una copia del bootloader GRUB (anche se ha il nome di default di UEFI, tipo '`BOOTx64.efi`').

Per controllare se la partizione EFI è montata usiamo il comando **mount**:

```
$ mount | grep efi

efivarfs on /sys/firmware/efi/efivars type efivarfs
(rw,nosuid,nodev,noexec,relatime)
/dev/sdb1 on /boot/efi type vfat (rw,relatime, ...)    <-- ECCOLA!
```

102.2 - Install a boot manager [2]

La partizione che contiene il boot loader deve essere etichettata con il flag **'boot'**.

Possiamo usare

- `# parted -l` oppure
- `# fdisk -l`

Per individuare sul nostro sistema tale partizione:

```
root@pve:~# parted -l

Model: ATA CT4000MX500SSD1 (scsi)
Disk /dev/sda: 4001GB
```

```
Sector size (logical/physical): 512B/4096B
```

```
Partition Table: gpt
```

```
Disk Flags:
```

Number	Start	End	Size	File system	Name	Flags
1	1049kB	2097kB	1049kB	fat32		
2	2097kB	540MB	538MB	fat32		boot, esp <--
ECCOLA!						
3	540MB	4001GB	4000GB	zfs		

Il flag '`boot`' può essere sempre attivato o disattivato con `parted` o `fdisk` come già visto per il flag '`esp`'.

Fatta questa premessa, studiamo le generalità dei principali boot loader.

GRUB Legacy (BIOS)

GRUB fino alla versione 0.97 è chiamato *GRUB Legacy*.

Posso installare GRUB alternativamente:

- `# grub-install /dev/sda` # all'interno del MBR del disco (primi 512kb) (obbligatorio)
- `# grub-install /dev/sda1` # su una partizione (in caso di chainloading)

Nei sistemi BIOS, è obbligatorio avere un boot loader installato nell'MBR del disco, perciò nel secondo caso si dovrà eseguire il cosiddetto *chainloading* ossia un *bootloader* scritto nell'MBR lancerà questo *bootloader* installato sulla partizione.

PRO TIP

Possiamo eseguire un **backup del MBR** prima di sovrascriverlo installando un nuovo boot loader:

- `# dd if=/dev/sda of=/path/to/backup-file count=1 bs=512`

In **GRUB Legacy** le voci di menù del boot loader sono memorizzate all'interno del file `/boot/grub/menu.lst`.

All'interno di questo file, le righe che iniziano con un cancelletto ('#') sono "commentate", ovvero sono voci non attive. È pratica comune per gli sviluppatori *commentare le impostazioni di default* per documentarne il comportamento predefinito. Rimuovendo il # di inizio riga, si rende attiva una impostazione ed è possibile cambiarne il valore di default.

Vediamo il significato delle principali direttive del file `/boot/grub/menu.lst`:

- `default 0` # indica la voce di menu selezionata di default ('0' è la prima voce)
- `timeout 5` # indica per quanti secondi la schermata di GRUB sarà visibile
- `title` # è il nome che compare nella voce di menu
- `hiddenmenu` # nasconde il menu (premere 'ESC' per renderlo visibile)

- `uuid` # definisce la partizione di boot (dove si trova il bootloader del SO)
- `initrd` # indica il percorso dove si trova l'initial ram disk o initramfs, se presente.
- `kernel /boot/... root=UUID ...` # indica il percorso dove si trova il kernel, la partizione da usare come radice (UUID) e altri parametri di lancio come visto nei capitoli precedenti

NOTA

Con GRUB Legacy è possibile modificare direttamente il file `menu.lst`, a differenza di GRUB2 dove questo file viene generato automaticamente con il comando `update-grub` e le impostazioni risiedono altrove (vediamo dopo).

RICORDA

puoi mostrare parametri che sono stati passati all'avvio del kernel in qualsiasi momento con il comando `$ cat /proc/cmdline`

Troubleshooting e recovery da GRUB

Dal menu GRUB posso digitare '`c`' per entrare in una shell minimale da cui posso, ad esempio:

- **Reinstallare GRUB sull'MBR:**
 - `'grub> root (hd0,0)'` # specifica la root del boot loader, in cui ci aspettiamo di trovare anche l'initrd
 - `'grub> setup (hd0)'` # installa grub sull'MBR del primo disco (conta da 0), oppure
 - `'grub> setup (hd0,0)'` # installa grub sulla prima partizione del primo disco
- **Avviare un SO non in elenco** inserendo manualmente i parametri: `'grub> root (hd0,0)'` `'grub> kernel /boot/vmlinuz root=/dev/sda1'` # specifica la root del kernel
`'grub> initrd /boot/initrd'` # specifica il percorso del ramdisk/initramfs
`'grub> boot'` # Avvia il SO

Come funziona GRUB 2

La versione **attualmente in uso** di GRUB, anche detta GRUB2, a differenza di GRUB Legacy, permette la gestione di molte più configurazioni, tra cui sistemi UEFI e dischi GPT.

A differenza di GRUB Legacy, in GRUB2 **non è più possibile modificare** `/boot/grub/menu.lst` direttamente:

```
#  
# DO NOT EDIT THIS FILE  
#  
# It is automatically generated by grub-mkconfig using templates  
# from /etc/grub.d and settings from /etc/default/grub  
#
```

Le impostazioni di GRUB2 risiedono in:

- `/etc/default/grub` # file di configurazione principale
- `/etc/grub.d/` # directory per file di configurazione "drop-in"

NOTA

Dopo aver modificato questi file di configurazione, è necessario **ri-generare il file menu.lst** con il comando

```
# grub-mkconfig -o /boot/grub/grub.cfg
```

Alcune distribuzioni mettono a disposizione un comando `update-grub` che è sostanzialmente un alias per il comando di cui sopra.

L'installazione di GRUB2, come per GRUB Legacy viene eseguita con `grub-install`:

- `# grub-install /dev/sda` # per i sistemi con MBR
- `# grub-install --target=x86_64-efi --efi-directory=/boot/efi --bootloader-id=GRUB` # per i sistemi EFI/GPT

PRO TIP

In GRUB2, oltre alla classica notazione `(hd0,0)` si può anche identificare il tipo di partizione, ad esempio '`(hd0,gpt1)`' oppure '`(hd0,msdos1)`'. In questo caso, la conta parte da 1.

ESERCIZIO

1. Esegui uno snapshot della macchina virtuale
2. Modifica le impostazioni di default di GRUB per mostrare i messaggi del kernel durante il boot al posto della splash screen.
3. Applica le modifiche e riavvia.
4. **Ha funzionato?** Sì/No? Perché? Dimenticato niente?

Troubleshooting - Menu di GRUB nascosto

A partire dalla *Ubuntu 18* (e numerose altre distro) **il menu di GRUB non viene visualizzato**, in quanto disabilitato nelle impostazioni in `/etc/default/grub`. Per abilitarlo, dobbiamo andare a cambiare le seguenti impostazioni:

- `GRUB_TIMEOUT_STYLE=hidden` --> `GRUB_TIMEOUT_STYLE=menu`
- `GRUB_TIMEOUT=0` --> `GRUB_TIMEOUT=5`

102.3 - Manage shared libraries [1]

Le librerie condivise, in **MS Windows** conosciute come '`dll`' o *Dynamic Linked Libraries*, sono dei **pezzi di software condivisi** tra più programmi per fornire funzionalità comuni. Ad es: il supporto al formato PDF (libreria `popppler`), ai formati multimediali (librerie `libav-*`) e via dicendo.

Static vs Dynamic linked libraries

La differenza è semplice:

- Se un programma **richiede librerie esterne** per funzionare (dipende quindi dalla presenza di tali librerie) viene definito **Dynamically Linked Executable**
- Se viceversa può essere eseguito **senza nessuna dipendenza** da librerie esterne (ossia contiene tutte le librerie necessarie al suo interno) viene chiamato **Statically Linked Executable**.

Naturalmente un programma compilato **staticamente** avrà una **dimensione maggiore** di un programma che fa uso di librerie esterne collegate dinamicamente. Quest'ultimo però, sarà intrinsecamente più prone a rotture in caso le librerie a cui si appoggia vengano aggiornate o modificate e smette di funzionare se vengono rimosse.

Quando tutto funziona come dovrebbe, le *librerie condivise* da cui dipende un programma sono **indicate come dipendenze** al gestore pacchetti e sono **installate automaticamente** con il programma in questione.

Come funziona il linking dinamico

Il comando '**l****dd**' ci mostra, per ogni eseguibile indicato come parametro, le librerie a cui si collega dinamicamente a tempo di esecuzione.

Ad esempio

- \$ **l****dd** /bin/ls

Mostra tutte le librerie di cui fa uso l'eseguibile di **ls**:

```
linux-vdso.so.1 (0x00007ffffd05c0000)
libsplash.so.1 => /lib/x86_64-linux-gnu/libsplash.so.1
(0x00007f122866b000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6
(0x00007f1228496000)
libpcre2-8.so.0 => /lib/x86_64-linux-gnu/libpcre2-8.so.0
(0x00007f12283fe000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2
(0x00007f12283f8000)
/lib64/ld-linux-x86-64.so.2 (0x00007f12286cb000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007f12283d6000)
```

Mentre non restituirà niente nel caso di un eseguibile compilato staticamente:

```
$ ldd /usr/bin/sln
 statically linked
```

Le librerie condivise hanno **estensione .so** (*shared object*) e all'avvio di un programma vengono **cercate nei seguenti percorsi**:

- `/lib/`
- `/usr/lib/`

Ma è possibile configurare il *dynamic loader* per **cercare anche in altri percorsi**:

- Modificando il file `/etc/ld.so.conf` oppure
- Inserendo file di configurazione "drop-in" all'interno di `/etc/ld.so.conf.d/`.

Alcuni percorsi aggiuntivi generalmente utilizzati sono:

- `/usr/local/lib/`
- `/opt/lib/`
- `/usr/lib64/`

IMPORTANTE

Dopo la modifica di `/etc/ld.so.conf` o `/etc/ld.so.conf.d/` è necessario **rigenerare la cache del dynamic linker** lanciando il comando `ldconfig`.

PRO TIP

Il comando '`ldconfig -p`' stampa a video l'elenco di tutte le librerie nella cache di `ldconfig` con relativo percorso.

La variabile LD_LIBRARY_PATH

Possiamo anche aggiungere uno o più percorsi **senza passare dalla cache di `ldconfig`** impostando/modificando la variabile `LD_LIBRARY_PATH`:

- `# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/morro/libs`

La modifica della variabile `LD_LIBRARY_PATH` non è permanente di default, ma ha **maggior priorità** della cache di `ldconfig`:

```
# PRIMA:
[morro@master ~]$ ldd /usr/bin/ls
    linux-vdso.so.1 (0x00007ffeaa9f6000)
    libcap.so.2 => /usr/lib/libcap.so.2 (0x00007fdcaa177c000)
<-- PRENDIAMO QUESTA COME ESEMPIO
    libc.so.6 => /usr/lib/libc.so.6 (0x00007fdcaa1533000)
    /lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
(0x00007fdcaa17d4000)

# MODIFICA DI LD_LIBRARY_PATH:
[morro@master ~]$ sudo cp /usr/lib/libcap.so.2 /home/morro/libs/
<-- CREO UNA COPIA
[morro@master ~]$ export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/morro/libs  <-- AGGIORNO
```

```
LD_LIBRARY_PATH

# DOPO:
[morro@master ~]$ ldd /usr/bin/ls
    linux-vdso.so.1 (0x00007ffd44fea000)
    libcap.so.2 => /home/morro/libs/libcap.so.2
(0x00007fc0ef82e000) <- MAGGIORE PRIORITÀ!
    libc.so.6 => /usr/lib/libc.so.6 (0x00007fc0ef5bf000)
    /lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-linux-x86-64.so.2
(0x00007fc0ef860000)
```

Questo può essere comodo per forzare momentaneamente il caricamento di una libreria personalizzata o ad una versione compatibile solo con il programma che si sta cercando di eseguire.

102.4 - Use Debian package management [3]

In GNU/Linux, il modo preferito per installare nuovi software è tramite un "gestore pacchetti" (package manager) che si occupa di **installare e mantenere aggiornati** tutti i software e relative dipendenze.

Sulle distribuzioni (distro) **Debian** e derivate i software installabili sono pacchettizzati in archivi con il formato **.deb** ed installabili tramite **dpkg**, **apt**, ecc... che vedremo a breve.

Questi pacchetti risiedono tutti in dei "magazzini centralizzati" detti **repository**. Su debian e derivate l'elenco dei repository si trova in **/etc/apt/sources.list** e **/etc/apt/sources.list.d/** (per le configurazioni drop-in).

Quando chiediamo al *package manager* di **installare un software**, questo si occuperà automaticamente di:

1. **Cercare il pacchetto** corrispondente sui *repository*
2. Ispezionare il pacchetto per **individuare tutte dipendenze**
3. **Scaricare** dai *repository* il pacchetto e le sue dipendenze
4. **Estrarre ed installare** il contenuto dei pacchetti scaricati nell'ordine corretto
5. Eseguire eventuali **script di post-installazione** previsti da ciascun pacchetto installato.

dpkg

dpkg non è un gestore pacchetti ma soltanto un **installatore** manuale di pacchetti.

Alcuni comandi utili:

- **# dpkg -i <pacchetto.deb>** # installa il pacchetto specificato (oppure no, se mancano le sue dipendenze)
- **# dpkg --info <pacchetto.deb>** # mostra informazioni sul pacchetto, tra cui le sue dipendenze

- `# dpkg --listfiles <nome-pacchetto>` # elenca i file contenuti nel pacchetto installato
- `# dpkg --contents <pacchetto.deb>` # elenca i file contenuti nel pacchetto .deb indicato
- `# dpkg -S </percorso/di/un/file>` # fornisce il nome del pacchetto da cui è stato estratto il file indicato (ricerca inversa)
- `# dpkg -l` # fornisce la lista di tutti i pacchetti installati sul sistema
- `# dpkg -l tz*` # filtra i pacchetti che iniziano con tz
- `# dpkg -r <nome-pacchetto>` # rimuove il pacchetto (mantenendo i file di configurazione installati)
- `# dpkg --purge <nome-pacchetto>` # elimina anche i file di configurazione

Il seguente comando, quando previsto dal pacchetto, procede alla sua riconfigurazione:

- `# dpkg-reconfigure <nome-pacchetto>`

Proviamo a capire da quale pacchetto (e quale versione) è collegato il comando '`java`':

- `# which java` # per vedere il percorso completo dell'eseguibile

E' probabile che il nome del comando sia in realtà un link simbolico ad un file, in questo caso un link a qualcosa del tipo `/etc/alternatives/java` per cui eseguo:

- `# update-alternatives --display java`

Per vedere il percorso di default collegato al comando '`java`'.

A questo punto posso digitare:

- `# dpkg -S <percorso-java>`

E capire a quale pacchetto appartiene il comando.

Leggere il `man dpkg` per tutte le opzioni e maggiori info sul comportamento.

apt-cache (legacy)

Questo tool esegue delle query sui pacchetti.

- `# apt-cache search <stringa>` # cerca pacchetti che hanno nel titolo o nella descrizione
- `# apt-cache showpkg <pacchetto>` # mostra informazioni sul pacchetto
- `# apt-cache stats` # mostra statistiche
- `# apt-cache unmet` # visualizza le dipendenze non soddisfatte
- `# apt-cache depends <pacchetto>` # mostra le dipendenze del pacchetto
- `# apt-cache rdepends <pacchetto>` # mostra le dipendenze inverse (i pacchetti che richiedono questo pacchetto)
- `# apt-cache pkgnames <stringa>` # mostra i pacchetti il cui nome inizia per

'apt' utilizza dei repository per fornirci tutte le informazioni di cui sopra, questi repository si trovano all'interno del file `/etc/apt/sources.list` e vengono aggiornati tramite il seguente comando:

- `# apt-get update`

E l'aggiornamento del sistema avviene tramite:

- `# apt-get upgrade`

apt-get (legacy)

È il "vecchio modo" per **interfacciarsi con il package manager** su debian e derivate.

NOTA

Più recentemente `apt-get` e `apt-cache` sono stati sostituiti dal comando `apt` che include le funzionalità di entrambi.

Alcuni esempi d'uso del comando '`apt-get`':

- `# apt-get install <pacchetto>` # installa anche le dipendenze
- `# apt-get -s install <pacchetto>` # simula l'installazione
- `# apt-get install -d <pacchetto>` # scarica solamente i pacchetti nella directory `/var/cache/apt/archives`
- `# apt-get clean` # elimina tutti i file dalla cache (pacchetti scaricati)
- `# apt-get remove <pacchetto>` # disinstalla il pacchetto
- `# apt-get autoremove <pacchetto>` # rimuove anche le dipendenze del pacchetto che non sono più necessarie
- `# apt-get autoremove --purge <pacchetto>` # rimuove anche i file di configurazione (comparirà un asterisco accanto al nome del pacchetto)

apt

In sostituzione di '`apt-cache`', '`apt-get`' e '`aptitude`' ora abbiamo il solo comando '`apt`', più semplice e completo:

- `# apt` # lanciato da solo da le informazioni sui principali utilizzi
- `# apt search <stringa>` # cerca tra i nomi dei pacchetti e le descrizioni
- `# apt show <pacchetto>` # mostra informazioni su un pacchetto
- `# apt install <pacchetto>` # installa un pacchetto e le sue dipendenze
- `# apt list <stringa>` # mostra i nomi dei pacchetti che rispettano i parametri di ricerca di
- `# apt update` # aggiorna la cache dai repository
- `# apt upgrade` # aggiorna il sistema
- `# apt full-upgrade` # a differenza di upgrade rimuove anche le versioni precedenti e le dipendenze non più richieste

ESERCIZIO

1. Utilizzando **apt**, cerca il pacchetto chiamato "sl" e mostra le sue generalità. Quali sono le sue dipendenze?
2. Installa sl ed eseguilo. Cosa succede?

QUIZ Se dovessi aggiornare il sistema e tutte le applicazioni installate tramite il package manager apt, quale sequenza di comandi sarebbe opportuno utilizzare?

ESERCIZIO (approfondimento)

Verifica a quali librerie dinamiche si aggancia l'eseguibile di **sl** a tempo di esecuzione.

Cosa rappresentano le voci in più?

Indizio: puoi usare **which** per ottenere il percorso completo di un eseguibile

102.5 - Use RPM and YUM package management [3]

Sulle distribuzioni **RedHat**, RedHat-like e derivate (**CentOS**, **Fedora**, **SUSE**, ...) si usano pacchetti in formato **.rpm**. I **repository** si trovano in **/etc/yum.repos.d/**. Generalmente un file per ciascun repository o gruppo di repository affine.

rpm

Il comando **rpm**, è il corrispettivo di **dpkg** per **Debian**, ovvero **non risolve le dipendenze** e non scarica i pacchetti, ma installa solo quelli già scaricati localmente.

Vediamo qualche esempio:

- **# rpm -qpi <pacchetto>** # (*q*uery (*p*)ackage (*i*nformations: informazioni sul pacchetto)
- **# rpm -qpl <pacchetto>** # (*q*uery (*p*)ackage (*l*ist: lista dei file contenuti nel pacchetto)
- **# rpm -i <pacchetto>** # *installa il pacchetto*
- **# rpm -qa** # (*q*uery (*a*ll: fa l'elenco di tutti i pacchetti installati)
- **# rpm -qi <pacchetto>** # (*q*uery (*i*nformations: informazioni sul pacchetto installato)
- **# rpm -ql <pacchetto>** # (*q*uery (*l*ist: lista dei file di un pacchetto installato)
- **# rpm -qf <percorso-file>** # (*q*uery (*f*ile: restituisce il pacchetto da cui è stato installato un file)
- **# rpm -e <pacchetto>** # (*e*rase: *disintalla un pacchetto*)

ESERCIZIO

1. Verifica quale pacchetto ha installato il file di configurazione **/etc/passwd**
2. Quali altri file sono stati installati dallo stesso pacchetto?

yum

È il gestore dei pacchetti delle distribuzioni Redhat-like. Più recentemente è stato **definitivamente sostituito da dnf** che, per la maggior parte dei casi, mantiene la stessa sintassi.

Vediamo qualche esempio:

- `# yum --help` # stampa del testo di aiuto
- `# yum search <stringa>` # cerca una stringa all'interno dei nomi e delle descrizioni dei pacchetti
- `# yum provides <file>` # ci dice quale pacchetto contengono il file
- `# yum info <pacchetto>` # restituisce informazioni sul pacchetto
- `# yum check-update` # come 'apt-get update'
- `# yum update` # aggiorna il sistema! Equivale ad un 'apt-get upgrade' su Debian
- `# yum list` # elenca tutti i pacchetti
- `# yum list <stringa>` # cerca i pacchetti con la stringa nel nome
- `# yum install <pacchetto>` # installa un pacchetto
- `# yum provides <percorso-file>` # restituisce da quale pacchetto è stato installato un file
- `# yum remove <pacchetto>` # disininstalla un pacchetto

Vi sono poi delle installazioni c.d. *di gruppo*, ossia che raggruppano più pacchetti per installare un intero ambiente, ad esempio un Desktop Manager oppure tutto il necessario per eseguire un Server Web:

- `# yum grouplist` # visualizza tutti i gruppi di applicazioni (es. "Server Web")
- `# yum groupinstall "Server Web"` # installa tutto il necessario per ottenere il risultato, in questo caso un Server Web

dnf

Dnf sostituisce definitivamente '`yum`' e, come detto, ha praticamente la stessa sintassi:

- `# dnf search <stringa>` # cerca una stringa all'interno del nome e della descrizione dei pacchetti
- `# dnf install <pacchetto>` # installa un pacchetto
- ...

Ma integra anche alcune novità come la possibilità di **scaricare soltanto** un pacchetto senza installarlo (precedentemente tramite `yumdownloader`) tramite:

- `dnf download <pacchetto>`
- `$ yumdownloader --resolve <pacchetto>` # scarica anche le dipendenze per poter (ad esempio) installare il pacchetto e tutte le dipendenze quando siamo offline

ESERCIZIO

1. Usando il gestore pacchetti `dnf`, individua il pacchetto che fornisce l'eseguibile `/usr/bin/ls`.
2. Più pacchetti presenti nei repository forniscono l'eseguibile specificato. Verifica quale di questi è installato.

QUIZ Se dovessi aggiornare il sistema e tutte le applicazioni installate tramite il package manager `dnf`, quale comando o sequenza di comandi sarebbe opportuno utilizzare?

ESERCIZIO GUIDATA

Scarica il pacchetto `setup`, estrailo e controlla il contenuto del file di configurazione `passwd` al suo interno:

1. `mkdir setup-pkg && cd setup-pkg`
2. `dnf download setup`
3. `rpm2cpio setup-*.rpm | cpio -dium`
4. `ls -l`
5. `cat etc/passwd` # nota la mancanza di "/" all'inizio di "etc".

102.6 - Linux as a virtualization guest [1]

Virtualizzazione vs Containerizzazione

Parliamo di **virtualizzazione** quando usiamo un **hypervisor**, per creare delle **macchine virtuali** che virtualizzano/emulano le periferiche di un computer. All'interno di una macchina virtuale è possibile installare un sistema operativo completo alla stessa maniera in cui lo installeremmo su un computer reale.

Un **hypervisor** è in grado di virtualizzare più sistemi operativi simultaneamente, finché ci sono risorse a sufficienza da riservare a ciascuna macchina virtuale. Ciascuna macchina virtuale esegue in totale isolamento dalle altre macchine virtuali o dal sistema "**Host**", a meno che non si configuri la rete in modalità bridge su un'interfaccia comune.

All'interno della macchina virtuale avviene tutto ciò che abbiamo discusso nei capitoli precedenti:

1. Viene lanciato il BIOS o UEFI
2. Viene caricato il boot loader
3. Viene lanciato il kernel
4. Il kernel avvia il sistema di init che completa l'avvio dei servizi.

Alcuni esempi di hypervisor: `vmware`, `virtualbox`, `linux-kvm`.

Parliamo di **Containerizzazione** quando usiamo un *Container Engine* o *Container Daemon* per eseguire un software e tutte le sue risorse in maniera isolata rispetto al resto del sistema operativo. Un processo all'interno di un container non è in grado di vedere o comunicare con i processi fuori dal container, ma il modello di isolamento è più blando e generalmente reputato meno sicuro rispetto a quello di un *hypervisor*.

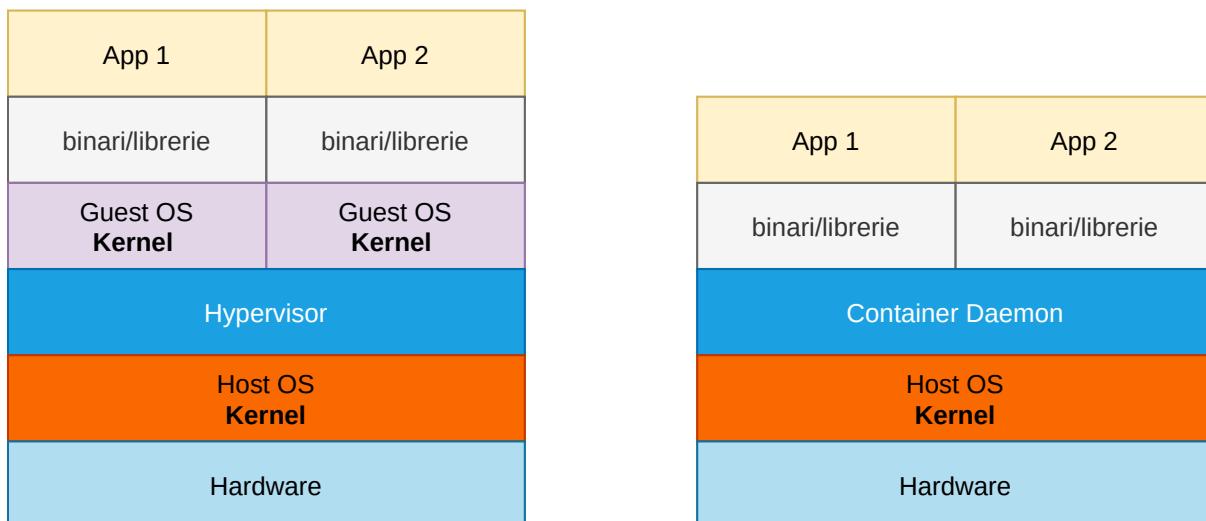
Un **container** è un ambiente minimale in cui è installato soltanto l'applicativo che si vuole eseguire e tutte le risorse di cui ha bisogno per funzionare correttamente, **ma non il kernel**. Il container non è mediato da un hypervisor ma *dal sistema operativo stesso*, perciò il kernel in uso sarà quello del sistema "host".

All'interno di un container:

1. Viene lanciato il sistema di init (nei container *LXC*), oppure
2. Viene lanciato direttamente il programma da eseguire (nel caso di *Docker* e *Podman*).

Virtualizzazione

vs

Containerizzazione

La virtualizzazione:

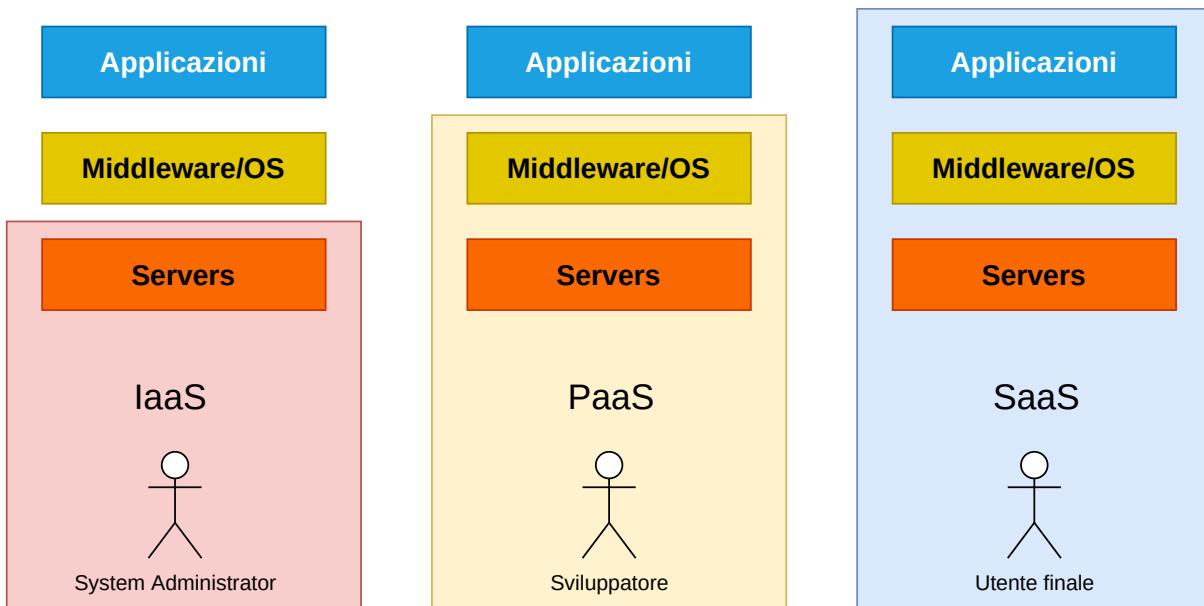
- Riserva staticamente le risorse all'avvio di ciascuna VM, a prescindere dall'uso effettivo
- Comporta un maggiore *overhead* dovuto alla virtualizzazione dell'hardware e la doppia indirezione del kernel
- Deve essere supportata dall'hardware fisico sottostante
- Ha un migliore isolamento ai fini della sicurezza informatica.

La containerizzazione:

- Riserva solamente le risorse effettivamente utilizzate dai servizi sottostanti
- Ha un minore *overhead* perché non avviene virtualizzazione ma solo allocazione di risorse
- Non permette l'esecuzione di un kernel all'interno di un container: viene usato quello dell'host
- Ha un isolamento potenzialmente meno sicuro

IaaS, PaaS, SaaS e terminologia Cloud

- **IaaS** : Infrastructure as a Service, provisioning hardware gestito da terzi, destinato ai sysadmin.
 - ES: Rackspace, Aruba, Linode, ...
- **PaaS** : Platform as a Service, Hardware e OS gestiti da terzi, destinato agli sviluppatori.
 - ES: IBM Cloud, Google Cloud, MS Azure, ...
- **SaaS** : Software as a Service, Applicativo pronto all'uso, destinato agli utenti finali.
 - ES: Gmail, Whatsapp, ...



Alcuni termini:

- **load balancing:** si frappone tra le richieste di accesso al servizio ed i server che lo erogano, bilanciando equamente il lavoro tra i server disponibili.
- **cloud computing instances:** sono le istanze in esecuzione simultanea dello stesso servizio (la citata 'scalabilità').
- **block storage:** astrae dai dettagli di implementazione sottostante e fornisce una interfaccia elementare per leggere e scrivere file. È possibile aumentare la capienza con un click senza preoccuparsi di come viene fatto al livello sottostante.
- **cloud networking** e software defined networking: Infrastruttura di rete virtuale scalabile trasparentemente. Al crescere del traffico posso richiedere un aumento di banda disponibile, senza preoccuparmi di come viene ottenuto il risultato.

Clonazione di macchine virtuali e accorgimenti

Una volta creata una macchina virtuale posso installare varie applicazioni e potrei volerla clonare tale e quale in modo da non dover ripetere le stesse configurazioni, oppure potrei volerla esportare come template per poterla utilizzare in altro contesto, magari con alcune configurazioni specifiche e diverse da quelle attuali.

clonazione

In caso di clonazione della macchina, devo fare attenzione ad alcuni aspetti che, post-clonazione, saranno da personalizzare sulla macchina clonata:

- il **MAC address delle schede di rete** che devo rigenerare;
- l'**UUID dei dischi** anch'esso da rigenerare (esistono varie utility tra cui '`tune2fs`');
- l'**hostname** che, se utilizzo le macchine clonate sulla stessa rete deve essere diverso;
- le **chiavi ssh dell'host** (vedi tutti file che iniziano con `/etc/ssh/ssh_host_*`), per la rigenerazione delle chiavi devo eseguire il comando '`ssh-keygen -A`' sulla macchina clonata;

- l'**UUID del dbus** anch'esso da rigenerare con il comando '`dbus-uuidgen --ensure`' che va a modificare il file `/var/lib/dbus/machine-id`;

esportazione

In caso di esportazione della macchina virtuale viene creato un file `.ova` che, in fase di importazione con Virtualbox permette di riconfigurare le caratteristiche hardware (GB di RAM, core della CPU, ...) come se si stesse creando la macchina virtuale.

cloud-init

Questo è uno strumento per personalizzare per istanze di cloud.

Ad esempio Canonical (Ubuntu) permette di creare uno script di personalizzazione che viene eseguito ad ogni clonazione di una macchina virtuale.

A esempio si può impostare che il sistema venga aggiornato all'ultima versione dei pacchetti (tramite `apt`), che vengano installati alcuni di essi e personalizzati alcuni file (come ad esempio l'hostname) in modo da facilitare le operazioni di post-clonazione citate sopra.

103 - GNU and Unix Commands.

103.1 - Work on the command line [4]

La shell è l'interprete dei comandi, anche conosciuto come "prompt dei comandi" su *MS Windows*.

Nell'ambiente shell è possibile eseguire tutti i comandi e gli applicativi "non grafici" ma anche eseguire comandi in base a delle condizioni, con il costrutto "*if*" o eseguire un blocco di comandi ripetutamente con un ciclo "*while*" e via dicendo.

In GNU/Linux esiste *più di una shell*. Diverse shell si differenziano principalmente nella sintassi dei costrutti sopracitati (cicli, condizioni, ecc...), mentre i **comandi/applicativi** che possiamo eseguire rimangono gli stessi, in quanto parte del sistema e non della specifica shell in uso.

In altre parole, la *shell* è una interfaccia per eseguire comandi e applicativi non grafici del sistema GNU/linux. Usare una *shell* diversa può comportare una *sintassi* diversa quando si usano costrutti condizionali e cicli, ma *nessuna differenza in termini di funzionalità*.

Stringendo ancora: **cambia la forma, non la sostanza**.

Cambiare shell

Per vedere su **quale shell sto usando**, posso consultare la variabile `$SHELL` che mi restituisce il percorso della shell in uso:

```
$ echo $SHELL  
/bin/bash
```

Ogni utente può impostare la propria shell predefinita con il seguente comando:

- `$ chsh -s </percorso/eseguibile/della/nuova/shell>`

L'**elenco completo** delle shell disponibili sul sistema è consultabile

- Con il comando `chsh -l`, oppure
- All'interno del file `/etc/shells`

L'utente root può anche impostare la shell di default di qualunque altro utente:

- `# chsh -s </percorso/eseguibile/della/nuova/shell> morro`

NOTA

La preferenza sulla shell di default è salvata in un campo del file `/etc/passwd` che contiene le generalità di ogni utente (ma non la password, contrariamente a quanto suggerirebbe il nome 😊)

È anche possibile cambiare shell 'al volo' ad esempio digitando `sh`, `bash`, `ksh`, e le altre shell disponibili.

ESERCIZIO

1. Imposta la shell predefinita del tuo utente per essere `sh`
2. Esegui un riavvio e verifica che la modifica sia persistente
3. Re-imposta la shell predefinita a bash, **senza l'uso di chsh**.

Entrare e uscire dalla shell

In linux abbiamo più modi per entrare ed uscire da una shell:

- Avviando un "emulatore di terminale" (konsole in KDE Plasma, GNOME Terminal, ...)
- Entrando in una TTY (CTRL + ALT + F1 per `tty1`, CTRL + ALT + F2 per `tty2`, ecc...)

Ogni TTY all'avvio richiederà il login, mentre l'esecuzione di più *emulatori di terminale* avviene nell'ambito della stessa sessione di login. Possiamo comunque effettuare un nuovo login con `su -l morro` anche dall'emulatore di terminale.

Digitando il comando '`ps`' le shell aperte sono individuate come rispettivamente:

- `pst` (pseudoterminal) se si tratta di emulatori di terminali;
- `tty` se sono console TTY.

Per uscire da una shell abbiamo le seguenti opzioni:

- Digitare **exit**;
- Premere **CTRL + D**;
- Digitare **logout** (per uscire da una tty o da una shell dove è stato fatto il login - esempio con **su - l**);

ESERCIZIO

1. Spostati su una tty ed effettua il login con il tuo utente
2. Lancia il comando **touch tty-experiment**
3. Chiudi la sessione tty e torna alla sessione grafica. Come si torna alla sessione grafica?

Le variabili nella shell bash

In ogni sessione esistono generalmente alcune variabili d'ambiente predefinite come **\$USER** oppure **\$PATH**.

Alcune di queste variabili sono *interne alla shell* (in questo caso bash), altre fanno parte dell'*ambiente* e sono disponibili a tutti i programmi lanciati dalla shell in oggetto.

Per visualizzare tutte le variabili d'ambiente uso il comando **env**:

```
root@morrolinux-2:~# env
SHELL=/bin/bash
LANGUAGE=en_US:en
PWD=/root
LOGNAME=root
XDG_SESSION_TYPE=tty
HOME=/root
LANG=en_US.UTF-8
XDG_SESSION_CLASS=user
TERM=xterm-256color
USER=root
...
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:
/usr/games:/usr/local/games:/snap/bin
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/0/bus
```

Per visualizzare **tutte le variabili** (comprese quelle **interne alla bash**) posso digitare **\$** seguito da un doppio **TAB**. Il completamento automatico suggerirà tutte le variabili impostate:

```
root@morrolinux-2:~# $<TAB><TAB>
...
$BASH_COMMAND          $HISTCMD          $MOTD_SHOWN
$SSH_CONNECTION        $HISTCONTROL      $OPTERR
$BASH_COMPLETION_VERSINFO
```

```
$SSH_TTY           $HISTFILE          $OPTIND
$BASH_LINENO       $HISTFILESIZE      $OSTYPE
$TERM              $HISTSIZE          $PATH
$BASHOPTS          ...
$UID
$BASHPID
$USER
...
```

Notare la presenza di variabili come `$HISTCMD`, `$HISTFILE`, `$HISTFILESIZE`, ecc... che definiscono il comportamento della history di bash. Queste sono a tutti gli effetti **impostazioni di bash** gestite tramite variabili "interne" e non esportate nell'ambiente.

Creare una variabile in bash è molto semplice:

- `<nome>=<contenuto>`

Ad esempio:

- `sito="https://morrolinux.it"`

Imposta il valore "`https://morrolinux.it`" all'interno della variabile "`sito`", creando la variabile se non esiste già.

Per visualizzarne il contenuto userò: `echo $sito`.

NOTA

Il segno di '\$' viene anteposto soltanto per **accedere al contenuto di una variabile** ma **NON** quando la si imposta.

```
> echo $sito
https://morrolinux.it

> bash

echo $sito

<vuoto>
```

La variabile così impostata è **visibile solo nella shell corrente** e non sarà visibile ai processi figli (i programmi lanciati da questa shell).

Per renderla visibile **anche ai figli** dobbiamo **esportarla**, operazione che può essere fatta anche in fase di assegnazione:

- `$ export <nome-variabile> # rende accessibile la variabile anche nelle shell figlie`

- `$ export <nome-variabile>=<contenuto-assegnato>` # esportazione di una variabile in fase di assegnazione

Per **rimuovere il valore assegnato** ad una variabile e renderla così 'vuota' utilizzo il seguente comando:

- `$ unset <nome-variabile>`

PRO TIP:

Possiamo mostrare l'**albero delle shell figlie** fino al terminale corrente con il comando:

- `$ ps --forest`

Altre opzioni della bash

La shell **sh** (e per eredità anche **bash**) ha alcune opzioni specifiche consultabili e modificabili tramite il comando:

- `$ set -o`

In particolare:

- **allexport** # definisce se ogni nuova variabile dovrebbe essere esportata automaticamente
- **noclobber** # definisce se impedire la sovrascrittura di un file esistente ad es. con gli operatori di redirezione

La variabile **\$SHELLOPTS** elenca tutte le opzioni di shell attualmente attive.

La shell **bash** ha anche un **ulteriore set di impostazioni** (non retrocompatibili con sh) consultabili tramite il comando **shopt**:

```
$ shopt
autocd          off
...
expand_aliases  on
histappend      on
histreedit      off
histverify      off
hostcomplete    off
...
```

Tra cui **expand_aliases** che definisce se gli alias configurati verranno utilizzati. **Ma cos'è un alias?**

Gli ALIAS in bash

Alcuni comandi danno il meglio soltanto quando sono usati con specifiche (e numerose opzioni). Lo scopo degli *alias* è quello di abbreviare l'uso di lunghi comandi con numerose opzioni.

Numerose distribuzioni fanno un uso massiccio di alias sui comandi più comuni. Ad esempio il comando `ls` per elencare file e directory nel percorso corrente, raramente è soltanto `ls`, provate voi stessi!

- `$ type ls` # ci dice che tipo di eseguibile è il comando 'ls'
- `$ alias ls` # ci dice quale alias corrisponde ad 'ls'
- `$ \ls` # esegue il comando 'ls' e non il suo alias
- `$ unalias ls` # elimina l'alias su 'ls' per la sessione corrente
- `$ alias ls='ls --color'` # definisce nuovamente l'alias 'ls' nella sessione corrente
- `$ alias` # mostra tutti gli alias attivi

Per rendere le modifiche permanenti queste devono essere inserite nel file di configurazione `~/.bashrc` che viene caricato ogni volta che faccio il login in una shell.

Per **disattivare l'uso degli alias** posso usare `shopt`:

- `$ shopt -u expand_aliases`

Per ri-attivare gli alias, invece:

- `$ shopt -s expand_aliases`

ESERCIZIO

Crea un alias "`ll`" per il comando `ls -lah`. Assicurati che sia **persistente**.

La bash history

L'elenco dei comandi digitati (*history*) si trova nel file: `~/.bash_history`. Tale file viene aggiornato **a fine sessione** ovvero, alla chiusura della shell, per impostazione predefinita.

Alcuni esempi di come sfruttare la history a nostro vantaggio:

- `$ history` # elenca gli ultimi comandi eseguiti, in ordine dal meno recente al più recente, numerati per riga.
- `$!20` # esegue nuovamente il 20-esimo comando della history
- `$!!` # esegue nuovamente l'ultimo comando lanciato
- `$!$` # corrisponde all'ultimo parametro usato nell'ultimo comando lanciato (utile quando si tratta di un percorso lungo, del nome di un file o di qualcosa di ripetitivo)
- `$!apt` # esegue nuovamente l'ultimo comando che iniziava con apt

PRO TIP

Per attivare o disattivare l'uso del carattere '`! !`' per l'esecuzione dalla history, agire sull'opzione '`histexpand`':

- `$ set +o histexpand` # per disattivare l'opzione
- `$ set -o histexpand` # per riattivarla

Posso anche **cercare nella history** con '**CTRL+r**' e iniziando a digitare parte del comando. La ricerca avviene a ritroso e si aggiorna man mano che digitiamo. Durante la ricerca possiamo premere:

- '**CTRL+r**' per cercare la successiva occorrenza a ritroso;
- '**INVIO**' per eseguire il comando trovato e visualizzato;
- '**ESC**' per uscire dalla ricerca ed eventualmente modificare il comando trovato prima di lanciarlo;
- '**CTRL+c**' per uscire dalla ricerca.

Per cancellare la cronologia eseguo il seguente comando:

- `$ history -c`

PRO TIP

Se `$HISTCONTROL` è impostata a 'ignoreboth' o 'ignorespace', i comandi che iniziano con spazio (es: ' `ls`') non verranno salvati dalla history! Utile per eseguire comandi che contengono dati sensibili o credenziali.

ESERCIZIO

Imposta la shell bash per non memorizzare in cronologia i comandi che iniziano con spazio. Assicurati che la modifica sia **persistente**.

Informazioni sul kernel: uname

Il comando '`uname`' fornisce informazioni sul kernel in uso.

- `$ uname --help` # chiarisce quali informazioni sono fornite a seconda delle opzioni utilizzate
- `$ uname -a` # fornisce praticamente tutte le informazioni
- `$ uname -r` # fornisce la versione del kernel

L'ultimo comando utile da espandere all'interno di qualche comando dove (appunto) è necessaria la versione del kernel in uso come nell'esempio seguente:

- `$ sudo apt install linux-headers-$(uname -r)`

Sarà espanso in:

- `$ sudo apt install linux-headers-5.4.0-125-generic`

O qualunque altra sia la versione del kernel installato e in esecuzione.

Ottenere aiuto: man e help

Approcciare un nuovo software da riga di comando può risultare scoraggiante perché non sappiamo *quali opzioni supporta* e con quale sintassi devono essere specificate queste opzioni.

Per questa ragione, la maggior parte dei comandi in GNU/Linux ha una pagina di manuale, generalmente molto completa e prolissa, accessibile tramite il comando `man`.

NOTA

Il comando `man` e relative pagine di manuale sono fornite dai pacchetti `man`, `man-db` e `manpages`. Per molte pagine del manuale è anche disponibile anche una **versione tradotta** fornita dal pacchetto `manpages` con il suffisso della propria lingua, ad esempio `manpages-it`.

Vediamo come si usa il comando `man`.

Ad esempio, `man ls` apre la seguente pagina:

```
LS(1)                               User Commands
LS(1)

NAME
    ls - list directory contents

SYNOPSIS
    ls [OPTION]... [FILE]...

DESCRIPTION
    List information about the FILEs (the current directory by
default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort
is specified.

    Mandatory arguments to long options are mandatory for short
options too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

    ...
```

Da qui, possiamo cercare qualcosa premendo `/` e digitando una stringa seguita dal tasto '**INVIO**' per andare al primo risultato. Passiamo al risultato successivo con '**n**' e al risultato precedente con '**N**'.

Per uscire da `man` premiamo '**q**'.

Se non sappiamo esattamente in quale pagina di manuale si trova l'argomento cercato, possiamo usare `apropos parola-chiave` per visualizzare tutte le pagine in cui la parola chiave cercata è menzionata:

```
$ apropos printf

...
printf (1)           - format and print data
printf (3)           - formatted output conversion
...
```

Da cui apprendiamo che esistono **due manuali** per *printf*, identificati dal numero tra parentesi.

Come mai?

Eseguiamo \$ **man 1 printf**:

```
PRINTF(1)                               User Commands
PRINTF(1)

NAME
    printf - format and print data

SYNOPSIS
    printf FORMAT [ARGUMENT]...
    printf OPTION

DESCRIPTION
    Print ARGUMENT(s) according to FORMAT, or execute according
    to OPTION:

    --help display this help and exit
    --version
        output version information and exit
    ...
```

e ora **man 3 printf**:

```
PRINTF(3)                               Linux Programmer's Manual
PRINTF(3)

NAME
    printf, fprintf, dprintf, sprintf, snprintf, vprintf,
    vfprintf, vdprintf,
    vsprintf, vsnprintf - formatted output conversion

SYNOPSIS
    #include <stdio.h>

    int printf(const char *format, ...);
```

```

int fprintf(FILE *stream, const char *format, ...);
int dprintf(int fd, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format,
...
...

```

Il primo è riferito al comando di shell "`printf`" mentre il secondo all'istruzione "`printf`" del linguaggio C!!!

Il manuale ha 8 sezioni, come descritto in `man manpages`:

1. User commands (Programs)
2. System calls
3. Library calls
4. Special files (devices)
5. File formats and configuration files
6. Games
7. Overview, conventions, and miscellaneous
8. System management commands

NOTA

I comandi sprovvisti di manuale, hanno generalmente almeno un'opzione `--help` per indicare all'utente quali opzioni accettano e con quale sintassi.

PRO TIP

- `$ which <comando>` *# fornisce il percorso all'eseguibile di un comando
- `$ whereis <comando>` # oltre all'eseguibile ci dice anche dove si trova il file di man (tipicamente `/usr/share/man/...`)

Quoting in bash

Il quoting è utile ogni volta che vogliamo usare caratteri che hanno un significato sintattico per l'interprete (bash) o viceversa.

I metodi di *quoting* sono 3 e sono intercambiabili a seconda dei casi e delle necessità:

- `\` : il carattere immediatamente successivo non venga interpretato dalla shell
- `'` : tutto ciò che è racchiuso tra *apici singoli* viene trattato come semplice stringa
- `"` : ciò che è racchiuso tra *doppi apici* è interpretato **solo in alcuni casi** (vediamo dopo)

Ad esempio, posso creare una directory con un nome che contiene degli spazi:

- `$ mkdir <nome>\ <directory>`

Lo spazio immediatamente successivo a `\` non viene interpretato come separatore di argomenti per `mkdir`.

I doppi apici ("") interpretano solamente i seguenti caratteri:

- '\$' # espansione di una variabile o di un comando (es. \$USER oppure \$(date))
- '' ' # un modo alternativo di espandere un comando (es. `date`)
- '\ ' # come già detto impedisce che il carattere successivo venga interpretato
- '!' # è il simbolo che permette l'espansione della history

L'uso dei doppi apici ("") non permette invece l'interpretazione tutti gli altri caratteri come ad esempio '><|&;:[{}]'.

Alcuni esempi su quanto visto sopra:

- \$ echo \$USER # stampa a video il contenuto della variabile user
- \$ echo "\$USER" # idem
- \$ echo '\$USER' # stampa a video la stringa \$USER
- \$ echo sono le ore \$(date +%T) # stampa a video "sono le ore hh:mm:ss"
- \$ echo `date` # come sopra
- \$ echo ``date`` # stampa a video la stringa `date`
- \$ echo "\$USER mi deve \$5" # stampa a video "utente mi deve "
- \$ echo "\$USER mi deve \\$5" # stampa a video "utente mi deve \$5"

Per approfondimenti:

- https://www.gnu.org/software/bash/manual/html_node/Quoting.html

103.2 - Process text streams using filters [2]

Su GNU/linux abbiamo moltissimi comandi e utility testuali, la vera potenza e flessibilità emerge soltanto quando siamo in grado di automatizzare i processi concatenando l'output di un comando come input di un secondo comando e via dicendo. Il potere espressivo è illimitato e per trarne vantaggio è sufficiente conoscere una manciata di utility per il processing del testo.

Tutte le utility illustrate di seguito supportano la lettura da `stdin` e l'output su `stdout`, perciò è possibile concatenarli per ottenere risultati incrementalmente in questo modo:

- `cat testo.txt | utility1 | utility2 | utility3 | ...`

dividere e unire file con `split` e `cat`

A volte può essere utile **dividere un file di testo** in più righe oppure **spezzare un file binario** in più parti per poi riunirlo in un secondo momento (ES: per inviarlo via email).

Per fare gli esempi utilizzeremo un file di testo creato con il seguente comando:

- `$ for l in $(seq 4500) ; do echo "riga numero $l" >> textfile.txt ; done`
- `$ wc -l textfile.txt # verifica che il file abbia 4500 righe`

Il comando `split` può essere usato per dividere un file in tanti piccoli file:

- `$ split textfile.txt` # suddivide il file in più file di 1000 righe ciascuno chiamati `xaa, xab, xac, ...`
- `$ wc -l x*` # conta il numero di linee (- l) in ogni file che inizia per `x`
- `$ split textfile.txt <nome.file>` # specifica il prefisso da utilizzare per i nomi dei file generati
- `$ split textfile.txt -l 100` # specifica di quante righe deve essere ogni file

Infine, possiamo **concatenare** nuovamente tutti i file parziali ordinati per nome con `cat`:

- `$ cat x*` # stampa a video il contenuto di ogni file, uno dopo l'altro
- `$ cat x* > file-unito.txt` # invece di stampare a video, scrive sul file `file-unito.txt` il risultato.

PRO TIP

Possiamo anche **spezzare un file binario** qualsiasi (archivio, video, audio o altro) con l'opzione `-b` eventualmente seguita dalla dimensione desiderata di ciascun segmento:

- `$ split -b 5m vmlinuz mybin.`

Se cerco di ottenere informazioni sul tipo di file per ogni file generato:

- `$ file *`

Posso osservare che **solo il primo file** della sequenza spezzata è identificabile, perché contiene l'*header* che descrive il tipo di contenuto del file. Gli altri pezzi sono porzioni dal centro e per questo non sono identificabili. I file così divisi non sono fruibili fino a quando non vengono nuovamente concatenati come visto sopra:

- `$ cat files-da-concatenare* > file-riunito`

cut

Molte utility in linux hanno un output in formato tabellare. Il comando '`cut`' "ritaglia" una o più **colonne** a scelta, con la possibilità di specificare il separatore.

- `$ cut -c2 <nome-file>` # visualizza la colonna 2
- `$ cut -c12-15 <nome-file>` # visualizza un range di colonne (dalla 12 alla 15)
- `$ cut -c12- <nome-file>` # visualizza tutte le colonne dalla 12 in poi
- `$ cut -c-11 <nome-file>` # visualizza tutte le colonne fino alla 11-esima.

Ad esempio, il file `/etc/passwd` contiene le generalità di ogni utente a sistema. Un utente per riga e ogni campo è separato dai due punti (`:`).

Vediamo:

- `$ cat /etc/passwd` # mostra il contenuto completo del file.

Per ottenere solo l'elenco degli utenti a sistema, mi basta stampare la prima colonna:

- `$ cut -d':' -f1 /etc/passwd` oppure

Che visualizza il primo campo (o *field*) con l'opzione '`-f1`' e specifica come delimitatore il relativo carattere '`:`'.

Posso ottenere lo stesso risultato concatenando l'output di `cat` come input per `cut` con l'operatore di pipe (`|`):

- `$ cat /etc/passwd | cut -d':' -f1`

Questo giochetto vale per qualsiasi comando che accetti un flusso dallo *standard input* e produca un output sullo *standard output*. Non c'è un limite a quante concatenazioni possiamo fare con l'operatore di pipe:

- `$ cat /etc/passwd | cut -d':' -f1 | sort`

Ad esempio, ordina (comando `sort`) alfabeticamente gli utenti estratti con `cut` al passaggio precedente!

Per estrarre più campi con `cut`:

- `$ cat /etc/passwd | cut -d':' -f1,7 # visualizza le colonne 1 e 7`
- `$ cat /etc/passwd | cut -d':' -f1-3 # visualizza le colonne 1,2,3`

NOTA

Se alcune righe hanno il carattere separatore ed altre no, quelle che ne sono sprovviste vengono visualizzate per intero.

Aggiungendo il parametro '`-s`' specifico che, qualora non esista il separatore del campo indicato, non venga stampata la relativa riga:

- `$ cat /etc/passwd | cut -d':' -s -f1,7`

L'opzione '`--complement`' inverte la selezione. Ad esempio:

- `$ cat <file> | cut -d':' --complement -s -f1,7`

Stampa tutti i campi **ad eccezione del campo 1 e 7**.

Infine posso anche specificare un mio delimitatore personalizzato da utilizzare in fase di output:

- `$ cut -d':' -s -f1,7 --output-delimiter=' #' <file>`

ESERCIZIO

Stampa a video i nomi dei moduli kernel attualmente caricati, eliminando eventuali altre colonne (dimensione, numero istanze, ecc..)

head e tail

I comandi **head** e **tail** **lavorano sulle righe** e sono essenziali per consultare i log su Linux: servono a stampare solo le prime o le ultime N righe di un file potenzialmente molto lungo.

Per vedere i primi messaggi (quelli più vecchi), la testa (*head*) del file:

- `$ sudo head /var/log/messages # vede le prime 10 righe di default`
- `$ sudo head /var/log/messages -n 5 # vede le prime 5 righe`

Alla stessa maniera '*tail*' mostra la fine (coda o *tail*) di un file::

- `$ sudo tail /var/log/messages # vede le ultime 10 righe di default`
- `$ sudo tail /var/log/messages -n 5 # vede le ultime 5 righe`

L'opzione *follow* aggiunta al comando '*tail*' permette di seguire in tempo reale l'aggiunta di nuove righe di log al file indicato:

- `$ sudo tail -f /var/log/messages # se il file non esiste da errore`
- `$ sudo tail -F /var/log/messages # segue il file anche se non esiste, avvisando`

Se passo al comando '*tail*' 2 file, il comando li segue entrambi:

- `$ tail -f <file-1> <file-2> # avvisandoci ogni volta quale file è stato modificato`
- `$ tail -f -q <file-1> <file-2> # l'opzione -q non specifica ogni volta quale file viene modificato ma visualizza solo la modifica`

ESERCIZIO

Individua il file di log del package manager in uso e stampa le ultime 50 righe.

ESERCIZIO

1. Apri il file di log del package manager in uso in modalità "follow"
2. Su un altro terminale, lancia l'installazione del pacchetto `cowsay` e osserva il log sulla prima finestra.

`grep`

Il comando `grep` lavora sulle righe. Può essere usato per **estrarre** o escludere da un flusso di testo **tutte le righe contenenti un pattern** o una corrispondenza esatta.

Alcuni esempi:

- `$ cat <text-file> | grep parola # estrae dallo stream di testo tutte le righe contenenti "parola".`
- `$ cat <text-file> | grep -i parola # idem, ma case insensitive: sia "Parola" che "parola".`
- `$ cat <text-file> | grep -w parol # whole-word: estrae le righe contenenti "parol" ma non "paro".`
- `$ cat <text-file> | grep -v parola # estra tutte le righe che non contengono "parola"`
- `$ cat <text-file> | grep -E <regex> # permette di specificare espressioni regolari estese.`

- `$ cat <text-file> | grep -A 5 parola` # stampa anche le 5 linee successive al match.

Naturalmente le opzioni molte di più e **combinabili** tra loro. Consulta `man grep` per maggiori dettagli.

NOTA: `grep` può lavorare su flussi di testo in streaming da pipe (`|`) o direttamente su un file specificato con questa sintassi:

- `grep -opzioni nomefile`

PRO TIP

Il comando `grep` ha un'opzione `-R` per cercare *ricorsivamente* un pattern su più file all'interno di una cartella specificata. In questo modo è possibile analizzare rapidamente un codebase o una directory contenente file di configurazione, alla ricerca di *voci specifiche* o *pattern* senza rincorrere ad un IDE avanzato.

ESERCIZIO

1. Stampa a schermo il contenuto di `/etc/passwd` filtrando soltando la riga relativa al tuo utente
2. Modifica la pipeline del punto 1 aggiungendo un filtraggio sul 7' campo del file `passwd` alla riga specificata.

ESERCIZIO

Stampa a schermo i nomi degli ultimi 3 pacchetti installati dal log del package manager.

SOLUZIONE (dnf)

```
grep -w "Installati:" /var/log/dnf.log | cut -d' ' -f4 | tail -n3
```

`tr` (o `translate`)

Il comando '`tr`' **lavora sui caratteri**. L'utilizzo principale è nella seguente forma:

- `$ cat <text-file> | tr '<SET1>' '<SET2>'`

Alcuni esempi:

- `$ cat <text-file> | tr [a] [b]` # sostituisce tutte le occorrenze di *a* minuscola con *b*
- `$ cat <text-file> | tr [a] [A]` # sostituisce tutte le occorrenze di *a* minuscola con *A* maiuscola
- `$ cat <text-file> | tr [:lower:] [:upper:]` # sostituisce tutti i caratteri minuscoli con caratteri maiuscoli
- `$ cat <text-file> | tr [:lower:] [:upper:] > new-file` # come sopra, ma salva il risultato su `new-file`
- `$ cat <text-file> | tr -d '.'` # sostituisce tutte le occorrenze di `'.'` con... niente.
Di fatto elimina i '.'

- `$ cat <text-file> | tr -d '.d' # elimina i due caratteri indicati '.' e 'd'`
- `$ cat <text-file> | tr -dc '.' # elimina tutto tranne il carattere indicato '.' (funzione complementare)`
- `$ cat <text-file> | tr -dc [:print:] # elimina tutti i caratteri non stampabili (compreso il carattere di acapo a fine riga)`

Per ottenere informazioni sul funzionamento di '`tr`' possiamo consultare:

- `$ tr --help`
- `$ info tr`
- `$ man tr`

WC

'`wc`' può essere usato per contare righe, parole e caratteri di un file o flusso di testo.

- `$ wc <file> # restituisce numero righe, parole e caratteri (byte)`
- `$ cat <file> | wc # come sopra, con l'uso della pipe`
- `$ wc -w <file> # restituisce il numero di parole`
- `$ wc -L <file> # restituisce il numero di caratteri di cui è composta la riga più lunga`
- `$ wc -c <file> # restituisce il numero di caratteri (o byte)`

ESERCIZIO

Utilizza `wc` per contare quanti moduli kernel sono attualmente caricati.

ESERCIZIO (avanzato)

Nell'esercizio precedente, il conto risultante da `wc` è riferito al *numero di righe* nell'output di `lsmod`, ma *la prima riga di lsmod contiene l'intestazione* e la conta risultante è sovrastimata di una unità. Combinando gli strumenti visti fin ora, **estrai il numero esatto di moduli caricati**.

SOLUZIONE

```
lsmod | tail -n +2 | wc -l
```

nl

'`nl`' può essere usato per aggiungere un contatore di riga ad un output testuale.

- `$ nl <file> # Aggiunge un contatore di riga sulla sinistra`
- `$ nl -i2 <file> #incrementa di 2 il contatore ogni riga (offset)`
- `$ nl -s. <file> #mette un punto ('.') come suffisso dopo il contatore`
- `$ nl -w2 <file> # cambia l'indentazione del numero di riga (default 6 spazi)`
- `$ nl -nrz <file> # zero-padding davanti alla numerazione`

sort e uniq

Qui si iniziano a intravedere le infinite possibilità di ricombinazione e concatenazione in pipe.

- `$ sort <file>` # ordinamento alfabetico delle righe
- `$ cat <file> | sort` # come sopra, usando la pipe
- `$ sort -r <file>` # ordinamento alfabetico inverso, da non confondere con
- `$ sort -R <file>` # ordinamento casuale
- `$ du -s ~/* | sort -n` # ordina per numero (-n) crescente (se il numero è a inizio riga)
- `$ sort -k 2,2 <file>` # ordina in base al secondo campo (campi separati da spazi)
- `$ sort -k 2 <file>` # ordina la riga a partire dal secondo campo in poi
- `$ cat <file> | sort | uniq` # mostra solo valori di riga unici (no duplicati consecutivi, ecco perché è preceduto da 'sort')
- `$ cat <file> | sort | uniq -d` # mostra solo le righe che presentano duplicati consecutivi
- `$ cat <file> | sort | uniq | wc -l` # conta i valori unici del file

ESERCIZIO

Servendoti del file di log del package manager della tua distribuzione, stampa a video l'elenco completo degli ultimi pacchetti installati, ordinati per nome, numerando ciascuna riga.

awk

Alcuni programmi testuali fanno output tabellare utilizzando come separatore il carattere di spazio. Questo rende più complesso il filtraggio per strumenti come `cut` che non sono pensati per gestire un numero arbitrario di caratteri nel separatore.

Ad esempio, se provassimo a stampare a video le prime due colonne dell'output di `lsmod`, ci accorgerebbero che `cut` non è lo strumento più indicato perché il separatore utilizzato è lo spazio, ripetuto diverse volte a seconda del padding necessario.

Un tool più completo e complesso che gestisce meglio questa casistica è `awk`, che può essere invocato in questo modo:

- `lsmod | awk '{print $1,$2}'`

In generale, bene evitare l'uso di `awk` quando non strettamente necessario in quanto notevolmente più lento rispetto agli altri tool presentati in questa sezione.

Awk è uno strumento estremamente completo e complesso, leggi il `man` relativo per maggiori dettagli sul suo utilizzo.

ESERCIZIO (avanzato)

Concatenando i comandi appresi in precedenza, stampa in un elenco ordinato in ordine decrescente i 3 moduli kernel con maggior numero di istanze attive.

SOLUZIONE

```
lsmod | awk '{print $1,$3}' | sort -k2 -n -r | head -n3
```

less

less può essere usato per **paginare l'output** di qualsiasi comando per poter **scorrere e cercare** nel suo contenuto. Ad esempio:

- `$ du -s ~/* | sort -n | less`

Paginerà l'output derivato dalla sequenza di comandi alla sua sinistra, lasciando la possibilità di scorrere e cercare parole chiave. Molto utile anche per **consultare file di log** molto lunghi o aprire file di testo in sola lettura.

less cattura tutto l'input della tastiera. **Per uscire premiamo q.**

paste

Lavora su righe e colonne di file e flussi di testo. Può essere usato per **trasporre** il testo da righe a colonne.

- `$ paste <file>` # come 'cat' visualizza il contenuto del file
- `$ paste -s <file>` # sopprime il carattere '\newline' visualizzando tutto su una riga e separando con una tabulazione le righe
- `$ paste -s <file> -d,` # separa le righe del file originale con il carattere ','

Quest'ultimo uso può tornare utile nelle conversioni di formato, ad esempio per trasformare un elenco di valori (uno per riga) in un elenco di colonne in formato CSV.

- `$ cat <file> | paste - - *` # trasforma una matrice di 1 colonna e N righe in 2xN/2 *
- `$ cat <file> | paste - - -` # trasforma una matrice 1xN in 3xN/3

od

od (Octal Dump) mostra i file in notazione ottale.

- `$ od -b <file>` # rappresentanza il file in ottale (octal dump)
- `$ od -x <file>` # rappresentazione in esadecimale

sed (intro)

Quanto segue è il riassunto di questa mini serie: [Corso base SED - Manipolare testo, input e output shell](#) che, seppur introduttiva, è anche fin troppo approfondita ai fini della certificazione in sé.

1: Sintassi delle espressioni Regolari

Le espressioni regolari ci permettono di individuare dei *pattern* nel testo, ad esempio:

- Sequenze numeriche con particolari caratteristiche (numeri di telefono, indirizzi IP, ...)
- Indirizzi email (NOTA: hanno sempre una chiocciola e un dominio di primo livello!)
- Date / datetime (quando espresse sempre nello stesso formato)

- ...

<https://regexr.com> è un sito dove ci si può esercitare con le espressioni regolari.

<https://regex101.com> è utile per il debug delle espressioni regolari

Ad esempio, per selezionare sia la parola '**color**' che '**colour**':

- '**color|colour**' # *il modo più semplice*
- '**colou?r**' # *il punto interrogativo significa zero o una corrispondenza del carattere che precede*
- '**colou*r**' # *l'asterisco vale se ci sono zero o più corrispondenze del carattere che precede pertanto selezionerà, oltre a color e colour, anche colouuuur*

Significato degli operatori:

- '**?'** # *zero o 1 corrispondenza del carattere che precede*
- '*****' # *zero o più corrispondenze del carattere che precede*
- '**+**' # *1 o più corrispondenze del carattere che precede (il carattere deve esserci almeno una volta!)*
- '**.**' # *è un jolly: seleziona qualsiasi carattere (anche lo spazio) tranne quello di acapo (\n)*
- '**{1,3}**' # *Specifica che il carattere che precede deve essere presente da 1 a 3 volte*

L'utilizzo delle parentesi quadre '**[]**' indica un insieme di caratteri in qualunque ordine:

- '**[.]**' # *in questo caso cerca il punto (i caratteri all'interno delle parentesi quadre non vengono interpretati)*
- '**[ciao]**' # *seleziona tutti i caratteri 'c', 'i', 'a', 'o' - troverà anche "caco".*
- '**[a-z]**' # *seleziona tutte le lettere minuscole*
- '**[A-Z]**' # *seleziona tutte le lettere maiuscole*
- '**[0-9]**' # *seleziona qualsiasi carattere numerico*
- '**[a-z] | [A-Z]**' # *seleziona tutte le lettere minuscole o le lettere maiuscole nota l'uso dell'OR: ' | '*
- '**[A-Z][a-z]+**' # *seleziona tutte le parole che iniziano con una maiuscola (senza ' | ' va in AND di default).*
- '**[^a]**' # *seleziona tutti i caratteri tranne 'a'*
- '**[^A-Z]**' # *seleziona tutti i caratteri tranne le lettere maiuscole*
- '**[^A-Z]**' # *seleziona tutti i caratteri tranne le lettere maiuscole e lo spazio*

Attenzione che alcuni simboli cambiano significato in base al contesto:

- '**^**' # *utilizzato senza parentesi quadre individua l'inizio della riga*
- '**^**' # *utilizzato dentro alle parentesi quadre agisce da operatore di negazione come visto sopra*

Quindi:

- '**^E**' # *seleziona il carattere 'E' se è a inizio riga*
- '**[^E]**' # *seleziona tutto ciò che NON è il carattere 'E'.*

- `'.'` # seleziona qualsiasi carattere che si trova a fine riga
- `'$'` # individua la **fine riga**
- `'[.]'` # seleziona il carattere '.' e il carattere '\$'

I pattern possono essere raggruppati in gruppi con l'uso delle parentesi tonde:

- `(pattern#1)|(pattern#2)` # seleziona il testo che corrisponde al pattern#1 oppure al pattern#2

2: SED - sostituire una parola

`'sed'` è uno *stream editor*, ossia individua dei pattern tramite le espressioni regolari e può modificare il contenuto a nostro piacimento.

Vediamo come sostituire una parola:

- `$ echo "giorno" | sed 's/giorno/notte/'`
- `$ echo "giorno 2" | sed 's/giorno/notte/'`
- `$ echo "buongiorno" | sed 's/giorno/notte/'`

Le espressioni regolari servono proprio per identificare esattamente la parola in modo da evitare, per esempio, che nel caso sia contenuta in altre parole si eviti di sostituirla.

Proseguiamo con gli esempi:

- `$ echo "buon giorno giorno" | sed 's/giorno/notte/'`

Sostituisce **solo la prima occorrenza**. Per continuare "globalmente" sul resto dello stream, aggiungiamo il flag `'g'` alla fine dell'espressione:

- `$ echo "buon giorno giorno" | sed 's/giorno/notte/g'`

Il comando `'sed'` ha quindi una sintassi strutturata come segue:

- `<comando>/<stringa-da-cercare>/[stringa-da-sostituire]/[parametri]`

Naturalmente posso redirigere un file direttamente nello standard input di sed:

- `$ sed 's/one/ONE/' < <file>`

3: SED - i separatori

sed usa il carattere "/" come separatore per i campi comando, ricerca e sostituzione. Per questa ragione, se vogliamo cercare o sostituire questo carattere con sed, dobbiamo prima farne l'escape con il carattere "" già visto in precedenza.

Ipotizzando di voler sostituire `/usr/bin/bash` con `/opt/bin/sh` all'interno di un file denominato `'percorsi'`:

- `$ sed 's/\usr\bin\bash/\opt\bin\sh/' < percorsi`

Questo che risulta di difficile lettura. In questi casi, possiamo usare un altro separatore per la sintassi di sed:

- `$ sed 's:/usr/bin/bash:/usr/bin/sh:' < percorsi`
- `$ sed 's|/usr/bin/bash|/usr/bin/sh|' < percorsi`

Per salvare il risultato in un '**nuovo-file**' posso redigere anche l'output di sed:

- `$ sed 's:/usr/bin/bash:/usr/bin/sh:' < percorsi > nuovo-file`

4: SED - aggiungere apici o parentesi

- `$ echo "ciao mamma" | sed s/'[a-z]*/(&)/g'`

Questa espressione mette tra parentesi tonde tutte le occorrenze incontrate:

- Il filtro `[a-z]*` cattura sono tutte le sequenze di lettere minuscole
- il simbolo `&` nella parte relativa alla sostituzione riprende l'occorrenza trovata

Per abilitare le *espressioni regolari estese* in sed devo usare il parametro `-r`:

- `$ echo "ciao mamma" | sed 's/[a-z]*(&)/g' # funziona senza problemi`
- `$ echo " ciao mamma" | sed 's/[a-z]*(&)/g' # racchiude l'occorrenza vuota perché abbiamo usato '*'`
- `$ echo " ciao mamma" | sed 's/[a-z]+(&)/g' # non funziona perché non abbiamo attivato le regexp estese`
- `$ echo " ciao mamma" | sed -r 's/[a-z]+(&)/g' # così funziona.`

Come si può notare, le parentesi tonde sono parte della sintassi di sed, ma possono comunque essere utilizzate all'interno dei campi di *ricerca o sostituzione* facendone l'escape: `'\('` e `'\)'`.

5: SED - duplicare un match

È sufficiente inserire due volte il riferimento al match nel blocco sostituzione:

- `$ echo "123 abc" | sed 's/[0-9]*/& &/'`

6: SED - scambiare l'ordine dei match

L'utilizzo delle parentesi, come detto in precedenza, crea un *gruppo* identificato da '**sed**' con un numero progressivo (da '**1**' per il primo pattern definito fino a '**9**' massimo) che viene richiamato nella parte della sostituzione con il numero preceduto dal simbolo di escape '`\`':

- `$ echo "123abc" | sed -r 's/([0-9]+)([a-z]+)/\2\1/' # inverte 123 e abc`
- `$ echo "123abc" | sed -r 's/([0-9]+)([a-z]+)/\2 \1/' # come sopra, ma aggiunge uno spazio`
- `$ echo "123 abc" | sed -r 's/([0-9]+).([a-z]+)/\2 \1/' # come sopra, se nel testo c'è uno spazio`

- ...

7: SED - eliminare numeri o testo o cambiarne l'ordine

È sufficiente stampare solo il gruppo (o i gruppi) che vogliamo includere nel risultato:

- `$ echo "abc123" | sed -r 's/([a-z]+).*/\1/' # stampa solo abc`
- `$ echo "abc123" | sed -r 's/([a-z]+)(.*)/\2/' # stampa solo 123`

8: SED - eliminare corrispondenze duplicate

Il riferimento al *gruppo* può anche essere usato **nel campo di ricerca**, ad esempio per trovare le parole doppie (inteso come due parole identiche separate da uno spazio) ed eliminarne una:

- `$ echo "ciao ciao a tutti" | sed -r 's/([a-z]+) \1/\1/'`

L'uso di `\1` subito dopo il primo gruppo di cattura ha la funzione di "duplicare" il risultato.

9: SED - tips and tricks

Prendiamo ad esempio `/etc/xattr.conf` che inizia con un blocco di righe commentate.

- `$ sed -n '1,5 p' /etc/xattr.conf # stampa le righe da 1 a 5 sopprimendo il resto dell'output (opzione '-n')`
- `$ sed -n '5,$ p' /etc/xattr.conf # come sopra ma dalla riga 5 alla fine del file`
- `$ cat /etc/xattr.conf | sed '/^#/d;' # stampa il file senza i commenti (righe che iniziano per '#')`
- `$ sed -i.bak '/^#/d;' myfile # ATTENZIONE! L'opzione '-i' modifica direttamente il file specificato. Se viene aggiunto un suffisso dopo '-i', verrà creata una copia del file originale con questo suffisso.`

`zcat`, `bzcat`, `xzcat`

Per visualizzare file di testo compressi non è necessario decomprimerli ma, a seconda del tipo di compressione, possiamo utilizzare il relativo comando che si comporterà esattamente come '`cat`' ma visualizzerà il contenuto del file compresso (utile ad esempio per visualizzare il contenuto dei log in `/var/log/..`):

- '`zcat`' visualizza file in formato '`*.gz`'
- '`bzcat`' visualizza file in formato '`*.bz2`'
- '`xzcat`' visualizza file in formato '`*.xz`'

`md5sum`, `sha256sum`, `sha512sum`

Per calcolare l'integrità di un file possiamo calcolare il suo hash e vedere se corrisponde con quanto dichiarato dal canale di distribuzione. L'*hash* è una rappresentazione alfanumerica di un

file calcolata tramite un algoritmo di hashing, tale per cui se anche un solo bit del file dovesse cambiare, l'hash risultante risulterebbe stravolto.

Possiamo calcolare diversi tipi di hash:

- `$ md5sum <nome-file> # obsoleto`
- `$ sha1sum <nome-file>`
- `$ sha256sum <nome-file>`
- `$ sha512sum <nome-file>`
- `...`

ESERCIZIO

1. Crea un file di testo chiamato "corsolinux.com" con il contenuto "<https://corsolinux.com>"
2. Calcola l'hash `md5` del file.
3. Rinomina il file. L'hash è cambiato?
4. Ora aggiungi un punto "." al termine della riga. L'hash è cambiato?

103.3 - Perform basic file management [4]

Copiare, spostare, creare, rinominare file da shell

Il comando `touch` aggiorna la data di accesso di un file ma è generalmente utilizzato anche per **creare dei file vuoti**:

- `$touch file # crea un file di testo vuoto chiamato "file"`

In bash, possiamo anche usare l'espansione con le parentesi graffe per creare più elementi simultaneamente:

- `$touch file{A,B,C} # crea , e`
- `$touch file{1..3} # per gli intervalli numerici. Crea , e .`

Il comando `mv` serve a **spostare e rinominare** file e cartelle.

- `mv fileA fileM # rinomina fileA in fileM`
- `mv fileA /nuovo/percorso/ # sposta fileA all'interno di /nuovo/percorso`
- `mv fileA /nuovo/percorso/fileM # sposta fileA all'interno di /nuovo/percorso rinominandolo in fileM`

Il comando `mkdir` serve a **creare una cartella** (o più):

- `$ mkdir cartella1 # crea `cartella1``
- `$ mkdir cartella1 cartella2 # crea cartella1 e cartella2 (nota lo spazio tra i 2 argomenti)`
- `$ mkdir dir{1,2,3} # come per touch`
- `$ mkdir dir{1...3} # come sopra`

- `$ mkdir -p <dir1>/<dir2>` # crea le cartelle che mancano nel percorso
- `$ rmdir <dir>` # elimina una cartella (solo se vuota)

Il comando `ls` elenca file e cartelle presenti in un percorso specificato:

- `$ ls /cartella` # mostra il contenuto di `/cartella`
- `$ ls -lah` # (*l*)ong listing, (*a*)ll e (*h*)uman readable: elenco dei file con dettagli.
- `$ ls -lhad <dir>` #'-d' permette di mostrare i dettagli della directory anziché il suo contenuto

ESERCIZIO

Crea un file vuoto chiamato "morrolinux" e una cartella con lo stesso nome. Che cosa accade?

SOLUZIONE

In Linux non è possibile creare un file con lo stesso nome di una directory perché le directory sono particolari tipi di file ma sono comunque dei file.

Il comando '`rm`' permette di eliminare file e directory:

- `$ rm nomefile` # elimina il file `nomefile`
- `$ rm -r <dir>` # elimina la directory e tutto il suo contenuto (`-r` = *recursive*)

Il comando `cp` serve a copiare file e cartelle e se necessario rinominare la nuova copia:

- `$ cp <file1> <file2>` # in caso esistesse già lo **sovrascrive senza avvisare**
- `$ cp -i <file1> <file2>` # (*i*nteractive, avvisa prima di sovrascrivere)
- `$ cp -ra <espressione> <destinazione>` # copia ricorsivamente (mantenendo i permessi), non segue i link simbolici
- `$ cp -ral <espressione> <destinazione>` # come il precedente ma segue anche i link simbolici

Visita il `man cp` per tutte le opzioni.

Archiviazione e compressione di file e cartelle

Prima di tutto facciamo la distinzione tra:

- **Archiviazione**: crea un file che contiene altri file, cartelle e sottocartelle.
- **Compressione**: riduce le dimensioni di un file.

Compressione con `gzip`:

- `$ gzip <file>` # trasforma nel formato compresso `<file.gz>` ed elimina
- `$ gzip -l <file.gz>` # mostra informazioni sulla compressione
- `$ gunzip <file.gz>` # decomprime ed elimina `<file.gz>`
- `$ zcat <file.gz>` # permette di leggere al volo file `gz` senza estrarli sul filesystem
- `$ gzip --help` # per tutte le opzioni

Compressione con **xz** (Algoritmo migliore di **gzip**):

- `$ xz -z <file>` # comprime il file in `<file.xz>` eliminando
- `$ xz -l <file.xz>` # mostra informazioni sulla compressione
- `$ xz -d <file.xz>` # per decomprimere (elimina il file compresso)

I comandi **gzip** e **xz** fa solo compressione, non archiviazione.

Il comando **tar** può fare sia **archiviazione** che **compressione** con diversi algoritmi:

- `$ tar -cvzf <archivio.gz> <cartella/espressione>` # (*c*)reate (*v*erbose) (*z*)compressione `gz` (*f*ile-name seguito dall'identificazione dei file da archiviare (se indicata una cartella la archivia/comprime con il suo contenuto))
- `$ tar -cvJf <archivio.xz> <cartella/espressione>` # come sopra, ma compressione `xz`
- `$ tar -cvf <archivio.tar> <cartella/espressione>` # crea solamente l'archivio senza eseguire nessuna compressione
- `$ tar -xvf <file.compresso> -C <cartella-destinazione>` # (-*x*) decomprime nella cartella specificata (-*C*). Se ometto la destinazione, decomprime nella cartella corrente

Alcune opzioni di '**tar**':

- '`-c`' # crea archivio
- '`-A`' # aggiunge un `<file1.tar>` ad un altro `<file.tar>`
- '`-r`' # aggiunge ad un `<file.tar>` dei file normali (non archivi)
- '`-u`' # inserisce nell'archivio una versione aggiornata di un
- '`-d`' # compara l'archivio con i file presenti su disco
- '`-t`' # lista i contenuti
- '`-x`' # estraе l'archivio
- '`-v`' # modalità verbosa
- '`-f`' # indica il nome del file da creare o decomprimere
- '`-J`' # comprime con 'xz'
- '`-z`' # comprime con 'gzip'

Leggi il **man tar** per tutte le opzioni.

**ESERCIZIO ** Crea un archivio in formato `.tar.xz` della cartella `/etc/` all'interno della tua `home`.

**ESERCIZIO **

1. Trova tutti i file con estensione `.tar.gz` sul filesystem
2. Elenca il contenuto dell'archivio più grande senza estrarlo

ESERCIZIO (avanzato)

Risvoli l'esercizio precedente in un unico passaggio combinando tutte le opzioni di filtraggio ed espansione necessarie in una pipeline.

SOLUZIONE

```
tar -tvf $(sudo find / -iname "*tar.gz" -exec ls -l {} \;
2>/dev/null|cut -d' ' -f5-|sort -nr|head -n1|cut -d' ' -f6)
```

rpm, cpio, rpm2cpio

'**cpio**' è un formato di archiviazione standard in linux ancora largamente utilizzato nei pacchetti '**rpm**' (RedHat/Fedora/CentOS e derivate).

- **\$ rpm2cpio <pacchetto.rpm> > <pacchetto.cpio>** # estrae da un pacchetto .rpm il relativo archivio in formato .cpio
- **\$ cpio -id < <archivio.cpio>** # estrae (-i) il contenuto dell'archivio nella directory corrente creando, se necessario, le relative directory (-d)
- **\$ cpio -t < <archivio.cpio>** # mostra il contenuto di un archivio .cpio
- **\$ rpm2cpio <pacchetto.rpm> | cpio -t** # mostra il contenuto del pacchetto .rpm
- **\$ ls <files-da-archiviare> | cpio -ov > archivio.cpio** # crea (-o) un archivio .cpio in modalità verbosa (-v)

Kilobit, Kilobyte e Kibibyte

I computer ragionano in modalità binaria:

- 1 bit può assumere solo due valori : 0 | 1
- 1 byte è una sequenza di 8 bits e può rappresentare 0 | 255 caratteri

E da qui le cose si fanno complicate.

C'è la notazione "fedele alla macchina" basata sui multipli del 2 con un nome che non piace a nessuno:

- 1 kibibyte = 2^{10} bytes = 1.024 bytes
- 1 mebibyte = 2^{20} bytes = $1.024 \times 1.024 = 1.048.576$ bytes
- 1 gibibyte = 2^{30} bytes = $1.024 \times 1.024 \times 1.024 = 1.073.741.824$ bytes

E c'è la notazione commerciale:

- 1 kilobyte = 1.000 bytes
- 1 megabyte = $1.000 \times 1.000 = 1.000.000$ bytes
- 1 gigabyte = $1.000 \times 1.000 \times 1.000 = 1.000.000.000$ bytes

E poi c'è la notazione preferita degli operatori telefonici: il kilobit, che è multiplo del bit e non del byte:

- 1 kilobit = 1.000 bit
- 1 megabit = $1.000 \times 1.000 = 1.000.000$ bit
- 1 gigabit = $1.000 \times 1.000 \times 1.000 = 1.000.000.000$ bit

Ricapitolando:

- 1Kb = 1 kilobit = 1.000 bit
- 1KB = 1 kilobyte = 1.000 bytes
- 1KiB = 1 kibibyte = 1.024 bytes

E via dicendo per Mb/MB/MiB, Gb/GB/Gib, ...

Block Size ('bs'): i *filesystem* generalmente scrivono su disco a blocchi di 4KiB (4.096 bytes).

TODO: RIVEDERE

Questo significa che, anche se il mio file è di dimensione inferiore, occuperà lo spazio minimo del blocco andando a "sprecare" spazio su disco. Questo però avviene per diminuire il numero di volte che viene effettuata la scrittura su disco in quanto file di piccole dimensioni sono sempre più rari e più cicli di scrittura significa un utilizzo maggiore della risorsa più preziosa, ovvero la CPU.

Facciamo un esempio, devo scrivere su disco 2 file:

- 1 file di 1K
- 1 file di 10M ((1.024K))

Lo posso fare su 2 *filesystem* diversi:

- il primo scrive a blocchi di 4K
- il secondo a blocchi di 1K

Sul primo *filesystem* i 2 files occuperanno uno spazio di 1.028K (4K per il file di 1K e 1.024K per il file di 1M) "sprecando" spazio su disco per 3K mentre sul secondo *filesystem* occuperà 1.025K, esattamente lo spazio necessario ai file in quanto il *block-size* è proprio di 1K pari al file più piccolo.

Tutto comporta che il primo *filesystem* necessiterà di 257 cicli di scrittura da 4K ciascuno (la dimensione del 'bs') mentre il secondo 1.025K cicli. Quindi la scelta è tra spreco di spazio (residuale all'effettiva grandezza dei singoli file) e spreco di risorse (CPU).

La dimensione minima del Block Size è pari a 512 bytes.

Il leggendario comando dd

Il comando **dd** serve a leggere e scrivere dati grezzi. Alcuni utilizzi possibili sono:

- Dump di interi dischi o partizioni su file e viceversa;
- Clonare dischi e partizioni;
- Convertire file di testo da uppercase a lowercase e viceversa;
- Sovrascrivere file con zeri o caratteri casuali per eliminarli in sicurezza;
- Fare backup del MBR di un disco;
- Misurare le prestazioni in lettura o scrittura di un disco.

Vediamo solamente alcuni esempi:

- \$ dd if=<lowercase> of=<uppercase> conv=ucase # tutto maiuscolo
- \$ dd if=/dev/zero of=<new-file> bs=4K count=1 # crea un file di 4KiB pieno di zeri
- # dd if=/dev/sda of=immagine.dd bs=4M # crea un'immagine disco completa di /dev/sda
- # dd if=/dev/sda of=/path/to/backup-file count=1 bs=512 # crea un backup dell'MBR del disco sda
- \$ dd if=/dev/zero of=/percorso/di/un/file bs=<dimensione_file> count=1 # Azzera il contenuto di un file per eliminarlo in modo sicuro.
- ...

ESERCIZIO

No. State alla larga da dd, è un bisturi affilatissimo.

find, locate, updatedb

Il comando 'locate' permette di cercare rapidamente tra i file presenti su disco da un indice generato in precedenza:

- \$ locate <nome-file-o-cartella> # cerca file o cartelle
- \$ sudo updatedb # aggiorna il db

Il problema di locate è che il db viene aggiornato periodicamente e pertanto potrebbe trovare un file che è appena stato eliminato o non trovare un file che è appena stato creato.

Per cercare in tempo reale senza indice usiamo il comando **find**:

- \$ find <percorso> -name <nome> # ricerca case sensitive
- \$ find <percorso> -iname <nome> # case insensitive
- \$ find <percorso> -iname '<espressione>' # cerca file e cartelle tramite espressione
- \$ find <percorso> -name <nome> 2>/dev/null # sopprime gli errori (tipo permesso negato)
- \$ find <percorso> -iname <nome> -exec ls -lhd {} \; # esegue il comando specificato su ogni singolo risultato della ricerca

Altri esempi di utilizzo del comando **find**:

- \$ find <percorso> -iname <file> -type f # cerca solo i file
- \$ find <percorso> -iname <file> -type d # cerca solo le cartelle
- \$ find <percorso> -maxdepth 1 -iname <file> # specifica la profondità massima in cui cercare in termini numeri di cartelle all'interno del percorso
- \$ find <percorso> -iname '*' -mtime +365 # cerca file più vecchi di 365 giorni
- \$ find <percorso> -iname '*' -mtime -30 # cerca file creati/modificati negli ultimi 30 giorni
- \$ find <percorso> -iname '*' -size +4k # cerca file di dimensione maggiore di 4 kilobytes

- `$ find <percorso> -iname '*' -size -4M # cerca file di dimensione minore di 4 Mega`
- `$ find <percorso> -user morro # trova i file che appartengono all'utente morro`
- `$ find <percorso> -group docenti # trova i file che appartengono al gruppo docenti`
- `$ find <percorso> -uid 1000 # trova i file che appartengono all'utente con UID 1000`
- `...`

ESERCIZIO

Trova tutti i file all'interno della home che sono stati modificati entro le ultime 24 ore.

ESERCIZIO

Trova tutti i file all'interno della home che sono stati modificati entro gli ultimi 5 minuti.

SOLUZIONE

```
find ~ -type f -mmin -5
```

bunzip2, unxz, gunzip

Le estensioni `.bz2`, `.xz` e `.gz` caratterizzano file compressi con i relativi algoritmi di compressione che possiamo decomprimere con i relativi tool:

- `$ bunzip2 <file.bz2>`
- `$ unxz <file.xz>`
- `$ gunzip <file.gz>`

Oppure possiamo utilizzare il comando `tar` per comprimere e decomprimere questi ed altri formati specificando l'opportuna opzione nella riga di comando.

103.4 - streams, redirection e pipe [4]

Operatori di redirezione e principali utilizzi:

Redirezione dell'output

L'operatore `>` è utilizzato per **redirigere l'output su file**. È possibile specificare diverse opzioni:

- `> file o 1> file` # redirige lo **standard output** (stream 1) su file, sovrascrivendo il contenuto se file esiste già.
- `>> file` # redirige lo **standard output** su file, appendendo in fondo al contenuto del file se esiste già.
- `2> file` # redirige lo **standard error** (stream 2) su file
- `&> file` # redirige **standard output** e **standard error** su file
- `2>&1` # redirige lo **standard error** sullo **standard output**
- `1>&2` # redirige lo **standard output** sullo **standard error**

NB: Nel caso dello standard output, non è obbligatorio specificare `1`:

- `2>&1 # redirige lo standard error sullo standard output`
- `>&2 1 # redirige lo standard output sullo standard error`
- `... 1`

Funzionano alla stessa maniera degli esempi sopra.

NOTA

L'opzione '`noclobber`' della shell bash impedisce la sovrascrittura di un file già esistente in caso di redirezione con singolo '`>`' generando un errore.

ESERCIZIO

1. Attiva l'opzione `noclobber` in maniera persistente sulla shell
2. Crea un file `corsolinux.com` con il contenuto del comando `date`.
3. Stampa a video con `cat` il contenuto del file appena creato
4. Esegui nuovamente `date` redirigendo l'output con l'operatore `>` nel medesimo file.
Qual'è il contenuto del file ora?

ESERCIZIO

Creare una **copia** del file `/etc/passwd` nella propria home **senza usare il comando cp**.

Redirezione dell'input

L'**operatore <** è utilizzato per redirigere il contenuto di un file sullo standard input del programma in esecuzione:

- `< file # legge il contenuto di file nello standard input del programma alla sinistra dell'operatore`

Ad esempio:

- `programma < file # redirige il contenuto di file sullo standard input di programma.`

ESERCIZIO

Creare una **copia** di `/etc/passwd` utilizzando **soltanto gli operatori di redirezione**.

SOLUZIONE

`> ciao.txt < text.txt`

L'**operatore <<** anche detto "Here Document" e può essere usato per redirigere sullo stdin del programma il contenuto di un "documento" testuale su più righe specificato "sul posto" (o in maniera interattiva da stdin).

È molto comodo ad esempio all'interno di script che "iniettano" sul filesystem uno o più file di configurazione.

La sintassi è la seguente:

```
<<TERMINATORE
contenuto
dell'here-document
su più righe no problem
TERMINATORE
```

Questo inietterà tutto il contenuto fino alla sequenza TERMINATORE esclusa nello stdin dell'hardware alla sinistra dell'operatore <<, o in un descrittore file specificato.

L'Operatore <<< è detto "Here String" ed è una variante non multi-riga dell'here-document, ma supporta l'espansione delle variabili e delle espressioni:

- `programma <<< "lunga stringa su singola riga" # legge il contenuto della stringa sullo stdin del programma`
- `programma1 <<< $(programma2) # legge l'output di programma2 sullo stdin di programma1`

Questo può essere utile anche per l'utilizzo in modalità non interattiva di alcuni programmi.

Ad esempio:

- `fdisk /dev/sdXY <<< $(printf "%\nnp\n\n\n\n\nw")`

Crea una nuova partizione sul block device specificato inviando al prompt del comando la sequenza:

- '`n`' # crea una nuova partizione
- '`\n`' # "INVIO" sul prompt per confermare il numero di partizione proposto
- '`p`' # specifica che si tratterà di una partizione primaria
- '`\n\n\n`' # "INVIO" x3 per confermare tutte le impostazioni di default proposte
- '`w`' # per scrivere le modifiche

L'operatore pipe

L'operatore pipe (`|`) può essere usato per redirigere lo standard output (`stdout`) del programma alla sinistra sull'input dello standard input (`stdin`) del programma di destra:

- `programma1 | programma2 # redirige lo stdout di programma1 nello stdin di programma2.`

In questo modo è possibile creare pipeline complesse concatenando gli effetti di ciascun software utilizzato. Particolamente utile per le operazioni di filtraggio dei flussi di testo.

Esempio di pipe:

- `$ ls -l | sort -k5 -nr # stampa l'elenco dei file ordinati per dimensione, dal più grande al più piccolo.`

PRO TIP

La pipe dopo il simbolo di redirezione dell'output (i.e. '`>|`') forza la sovrascrittura del file esistente anche se è attiva l'opzione '`noclobber`'.

Il comando tee

Quando si fa redirezione dello `stdout` su file con l'operatore `>`, l'output della shell non è più visibile a terminale.

Il comando `tee` può essere usato per "duplicare" l'output sia su file che su `stdout`:

- `cat file1.txt | wc -l | tee file2.txt`

stampa a schermo l'output di `wc -l` e contemporaneamente lo scrive nel file `file2.txt`

PRO TIP

Utilizzando una shell da utente non privilegiato, non è possibile fare redirezione dell'output con privilegi elevati, nemmeno utilizzando `sudo`:

- `echo "127.0.0.1 google.com" >> /etc/hosts # non funziona: non abbiamo i permessi per scrivere in /etc/.`
- `sudo echo "127.0.0.1 google.com" >> /etc/hosts # nemmeno: >/>`
sono eseguiti con i privilegi standard. **Una possibile soluzione** è usare `tee` in questo modo:
- `echo "127.0.0.1 google.com" | sudo tee -a /etc/hosts # FUNZIONA!`
Nota che stiamo usando `sudo` alla destra della pipe, in questo modo solo `tee` esegue con privilegi elevati.

Il comando xargs

`xargs` Prende in ingresso una serie di elementi separati da spazi o `\newline` e li passa ad un programma specificato a gruppi di `-n` elementi:

- `$ echo {1..9}`
- `$ echo {1..9} | xargs -n1`
- `$ echo {1..9} | xargs -n3`
- `$ echo "1 2 3" | xargs -n3 echo "N"`

Molto più frequente è l'uso di `xargs` per **invocare un programma con una lista di argomenti**.

Ad esempio, possiamo creare un archivio con tutti i file trovati dal comando `find` passando i risultati di `find` come **parametri di invocazione per tar**.

- `$ find <percorso> -iname '<espressione>' | xargs tar -czvf archivio.tar.gz`

Invoca `tar` appendendo ai parametri di invocazione tutti i risultati del comando `find`.

Un'opzione di **find** particolarmente utile quando usato in combinazione con programmi esterni come **xargs** è **-print0**, che aggiunge un carattere di terminazione al posto del **newline** di fine riga, permettendo di interpretare correttamente i percorsi contenenti caratteri di newline o simili.

Dal manuale di **find**:

```
-print0
    True; print the full file name on the standard output,
    followed by a null character (instead of the newline character
    that -print uses). This allows file names that contain
    newlines or other types of white space to be correctly inter-
    preted by programs that process the find output. This
    option corresponds to the -0 option of xargs.
```

103.5 - Create, monitor and kill processes [4]

I jobs in Linux

Quando un comando esegue per lungo tempo, tenendo impegnato il prompt della shell, posso premere:

- '**CTRL-c**' # interrompe il processo mandandogli il segnale **SIGINT**
- '**CTRL-z**' # mette in pausa il processo (segnale **SIGTSTP**). Al processo sarà assegnato un numero di 'job' progressivo (partendo da '1')

A questo punto:

- **\$ jobs** # elenca i jobs gestiti dalla shell e il loro stato (in esecuzione o stoppato)
- **\$ bg 1** # riprende l'esecuzione del job 1 lasciandolo in background
- **\$ fg 1** # riprende l'esecuzione del job 1 riportandolo in foreground

Posso anche decidere di lanciare un comando e metterlo subito in esecuzione in background aggiungendo il simbolo '**&**' alla fine della riga di comando.

- **\$ programma &** # lancia **programma** in background restituendo il numero di job creato

NOTA

Se non redirigo **stdout** e **stderr** dei comandi che metto in background, questi continueranno ad essere stampati sulla shell che li ha lanciati! E' pertanto consigliabile redirigerli in qualche file oppure verso '**/dev/null**' quando si mette in background un comando:

- **programma > /dev/null &**

Eseguire processi con nohup

Normalmente, la chiusura della shell da cui lancio un comando, termina anche i processi figli lanciati da quella shell, anche se sono in background e non hanno finito di eseguire.

Per eseguire un comando in modo che non venga terminato alla chiusura/disconnessione dalla shell o dal sistema, posso utilizzare il comando '**nohup**' :

- `$ nohup comando &`

Esegue il programma in background redirigendo lo **stdout** e **stderr** ad un file chiamato **nohup.out** nella cartella corrente, in modo da poter consultare l'output generato anche in un secondo momento.

ps e **pgrep**

Il comando **ps**

Restituisce l'elenco dei processi in esecuzione.

- `$ ps` # mostra i processi in esecuzione sulla shell attuale: (bash, ps ed eventuali processi figli in background)
- `$ ps -lf` # restituisce un elenco più dettagliato
- `$ ps -elf` # restituisce un elenco dettagliato di **tutti i processi in esecuzione**, non solo su questa shell.
- `$ ps -u nome-utente` # restituisce tutti processi eseguiti da nome-utente

NOTA

Il valore di '**nice**' (NI) può assumere valori da -20 a +19:

- '**-20**' # significa che il processo è 'poco gentile' (nice) e si prende molto tempo sulla cpu
- '**+19**' # significa che il processo è 'più gentile', lasciando più tempo di CPU agli altri processi

Il comando '**pgrep**'

Restituisce tutti i PID (Process ID) dei processi in esecuzione con il nome specificato.

- `$ pgrep sync` # restituisce i pid dei processi che **contengono** sync (ad.es: "sync", "syncall", "syncthing", ...)

Lo stesso risultato può essere ottenuto con

- `$ ps -e | grep sync`

POSIX SIGNALS

I segnali (o signals) sono uno standard POSIX per inviare notifiche ai thread o processi in esecuzione. Sono ancora comunemente usati in GNU/Linux e MacOs per notificare ai processi

determinati eventi.

Ogni segnale ha un codice identificativo a cui corrisponde un nome:

- 1) **SIGHUP** # *forza il reload del processo*
- 2) **SIGINT** # *chiede l'interruzione del processo (CTRL+C dalla shell padre manda questo segnale)* - *il programma può decidere di ignorare questo segnale... prova con `st`*
- 15) **SIGTERM** # *'termina' il processo in modo regolare (segnale di default di `kill`)*
- 9) **SIGKILL** # *'uccide' il processo brutalmente (si usa generalmente se non funziona il precedente)*
- ...

ESERCIZIO

Individua la pagina di manuale che descrive tutti i possibili segnali e relativa descrizione.

`kill`, `pkill` e `killall`

Il comando **kill** (e similmente **pkill**) può essere usato per inviare un segnale ad un processo specificato.

- `$ kill <numero-pid>` # *manda al processo corrispondente il segnale 15 (SIGTERM)*
- `$ kill -9 <numero-pid>` # *manda al processo corrispondente il segnale 9 (SIGKILL)*
- `$ pkill <nome-processo>` # *manda SIGTERM a tutti i processi che contengono nel nome (pericoloso!)*
- `$ killall <nome-processo>` # *come il precedente, ma richiede il nome esatto del processo (già meglio)*
- `$ killall -u morro` # *termina tutti i processi eseguiti dall'utente morro*

ESERCIZIO [1/2]

1. Lancia il comando `ping corsolinux.com` facendo in modo che non termini alla chiusura del terminale.
2. Chiudi la finestra di terminale.
3. Apri una nuova finestra di terminale e verifica se ci sono job presenti. Perché?

ESERCIZIO [2/2]

1. Verifica se il processo di ping è ancora in esecuzione
2. Segui il log generato da nohup. Sta funzionando?
3. Termina il processo di ping e verifica che sia effettivamente terminato.

`top`, `free`, `uptime`

Il comando '**top**' mostra parecchie informazioni sul sistema, tempo di attività processi in esecuzione.

Lanciato senza alcun parametro esegue in modo interattivo:

- '**l**' # abilita/disabilita i dati sull'uptime
- '**m**' # abilita/disabilita le informazioni sulla memoria
- '**t**' # abilita/disabilita le informazioni sull'uso cpu
- '**F**' # entra nelle impostazioni di Fields Management

All'interno di *Field Management* posso premere:

- '**d**' oppure '**SPACE**' # aggiunge toglie la colonna da visualizzare
- '**s**' # su una determinata colonna ordina i processi sulla base di quella colonna
- '**ENTER**' # torna alla visualizzazione interattiva di 'top' salvando le modifiche

Per invertire l'ordinamento predefinito basta premere il tasto '**R**' dalla in modalità interattiva.

Il comando '**top**' può anche essere eseguito in modalità *batch* ossia non interattiva, tramite l'utilizzo del parametro '**-b**':

- `$ top -b -n2 -d5` # esegue 'top' 2 volte a distanza di 5 secondi

Due comandi utili che forniscono alcune informazioni reperibili anche con '**top**':

- `$ uptime` # restituisce la prima riga di '**top**'
- `$ free -m` # restituisce informazioni sull'uso della memoria come in '**top**'

ESERCIZIO

1. Esegui il seguente comando: `dd if=/dev/zero of=/dev/null bs=1M &`
2. Individua il **PID** del processo dd. Quanto tempo di CPU sta consumando?
3. Termina il processo.

La cartella proc

La cartella `/proc` è un filesystem virtuale che contiene una rappresentazione "a file" dei processi in esecuzione.

All'interno di `/proc` troviamo tante sottocartelle quanti sono i processi in esecuzione. Ogni cartella ha come nome il pid del processo corrispondente e contiene molte informazioni utili sul processo stesso.

Le applicazioni di monitoraggio dei processi (come ad esempio '**top**') attingono proprio a questa cartella per mostrare tutte le informazioni sui processi attivi.

ESERCIZIO

1. Esegui il seguente comando `ping -R morrolinux.it >> /dev/null &`
2. Individua il processo in esecuzione nella cartella `/proc`.
3. Stampa a schermo i parametri con cui è stato invocato il processo. \
4. Termina il processo senza usare `kill`. HINT: "jobs".

ESERCIZIO (avanzato)

Esegui l'esercizio precedente risolvendo i punti 2 e 3 con una unico comando.

SOLUZIONE

```
cat /proc/$(pgrep ping)/cmdline
```

watch : tenere monitorati i comandi

Il comando '**watch**' è una utility che esegue periodicamente (default ogni 2 secondi) un comando specificato come parametro:

- `$ watch -n 1 ls -lh /var/log` # esegue il comando 'ls' ad intervalli di 1 secondo
- `$ watch -n 5 ping -c 1 google.com` # esegue 1 ping a google ogni 5 secondi

Può tornare molto utile per monitorare il contenuto di una cartella, il peso di un file in fase di modifica e via dicendo.

tmux moltiplicatore di terminali

Il comando '**tmux**' aggira i limiti che si hanno generalmente quando ci si collega tramite terminale remoto ('**ssh**') o quando si utilizza un solo terminale e si deve gestire tutto da una unica finestra.

Inoltre '**tmux**' permette anche di gestire delle sessioni di lavoro che possono essere lasciate attive nella macchina anche dopo il nostro logout fino al successivo collegamento (similmente a **nohup**).

All'avvio di '**tmux**' vediamo una barra in basso che gestisce tutte le '**shell**' attive.

L'interazione con il programma avviene tramite la combinazione **CTRL-b** seguita dal carattere che identifica il comando che vogliamo eseguire, vediamo i principali:

- '**c**' # crea una nuova shell bash (o finestra)
- '**,**' # rinomina la finestra corrente (contrassegnata con '*')
- '**p**' # si sposta alla finestra precedente - (**p**)revious
- '**n**' # si sposta alla finestra successiva - (**n**)ext
- '**w**' # crea un elenco delle finestre dove si può selezionare quella dove ci si vuole spostare
- '**%**' # divide verticalmente la finestra corrente
- '**"**' # divide orizzontalmente la finestra corrente
- '**t**' # mostra un blocco schermo con l'orario. Per sbloccare basta premere INVIO.
- '**Up/Down/Left/Right**' # si muove tra le varie shell create nella finestra corrente
- '**d**' # si scollega dalla sessione corrente **lasciandola attiva sulla macchina** - (**d**)etach
- '**CTRL-d**' # non preceduta da 'CTRL-b' equivale a uscire dalle varie shell con 'exit'

Dopo essersi scollegati con '**d**' la sessione rimane attiva sulla macchina anche se ci scolleghiamo dalla macchina stessa. Una volta che torniamo ad effettuare il login ci possiamo ricollegare:

- `$ tmux ls` # elenca le sessioni attive sulla macchina
- `$ tmux attach -t <nome sessione>` # si collega ad una sessione
- `$ tmux kill-session -t <nome sessione>` # termina la sessione specificata.

Una nuova sessione può essere creata con il semplice comando '`tmux`' e gli può essere assegnato un nome tramite il parametro '`-s`':

- `$ tmux new -s <nome-sessione>`

103.6 - Modify process execution priorities [2]

`nice`, `renice` e `priority`

Indipendentemente da quanto sia potente un calcolatore, ci saranno sempre **più processi da eseguire in parallelo di quanti core e thread abbiamo a disposizione**.

Per questa ragione, i processi devono **fare a turni per eseguire sulla CPU**. Ad ogni processo è concesso di eseguire sulla CPU per una porzione di tempo inversamente proporzionale all'affollamento.

Quando ad un processo è concesso più tempo di CPU rispetto agli altri, si dice che esegue con **maggior priorità**.

Viceversa, si dice che esegue con **minore priorità**.

In GNU/Linux, possiamo modificare il tempo concesso a ciascun processo con il parametro **NICE**.

`Nice` indica quanto un processo è "carino" nei confronti degli altri.

- Un processo *più carino*, cederà agli altri processi più fette del suo tempo di CPU.
- Un processo *meno carino*, chiederà agli altri *più tempo per sè*.

NICE può assumere i valori da:

```
-20 .. 0 .. +19
^
DEFAULT
```

La **priorità** di un processo è quindi influenzata dal suo valore di **nice**:

- Un valore di *NICE alto* = *bassa priorità*
- Un valore di *NICE basso* = *alta priorità*.

I valori di **priorità** e **nice** di ciascun processo sono riportati in `top`.

Alternativamente, possiamo usare:

- `$ ps -l`

Per visualizzare l'elenco dei processi comprese le colonne relative alla priorità (**PRI**) e al valore di nice (**NI**):

```
root@pve:~# ps -l
F S   UID      PID      PPID     C PRI  NI ADDR SZ WCHAN TTY
TIME CMD
4 S     0      5099      4393     0  80    0 - 1369 do_sel pts/0
00:00:00 agetty
4 S     0     121137      2  0  60 -20 -      0 taskq_ 16:39
00:00:00 [z_trim_iss]
...
```

NOTA

Nell'output di **ps** il valore di default di **PRI** è calcolato come **80 + NI**.

Per eseguire un programma con una priorità differente, usiamo il comando **nice**:

- **\$ nice -n 5 <comando>** # Lancia un comando con valore di nice 5 e quindi priority = 85.

Per modificare la priorità di un programma in esecuzione usiamo **renice**:

- **\$ renice -n 7 <PID>** # cambia il valore di nice di un processo.

NOTA

Solo l'utente **root** può assegnare ad un processo un valore di nice **negativo**.

Gli utenti standard possono cambiare il valore di **nice** da 0 a +19.

Similmente, possiamo anche cambiare la niceness di più processi contemporaneamente:

- **\$ renice -n 0 -u morro** # imposta la niceness dei processi di morro a 0.

ESERCIZIO (avanzato)

1. Esegui questo comando **dd if=/dev/zero of=/dev/null &** tante volte quanti sono i core disponibili +1.
2. La CPU salterà al 100% di utilizzo su tutti i core.
3. Modifica la priorità del primo processo assegnando valore di **nice = -20**.
4. Attendi un minuto e confronta lo stato del processo con NICE=-20 con i restanti.
5. Noti qualche differenza?

HINT: Per leggere lo stato di un process **dd** in esecuzione, invia un segnale **SIGUSR1** al suo PID.

103.7 - Search text files using regular expressions [3]

Le espressioni regolari (regex)

Come visto in precedenza, le espressioni regolari sono un potente strumento per filtrare del testo definendo delle regole in maniera piuttosto flessibile.

In GNU/Linux, possiamo trarre vantaggio da questa potenza espressiva con **grep** e **sed** tra le altre utility.

Alcuni esempi con grep:

- `$ grep morro <file>` # restituisce tutte le linee che contengono 'morro', **case sensitive**
- `$ grep -i morro <file>` # restituisce tutte le linee che contengono 'morro', **case insensitive**
- `$ grep '[Mm]orro' <file>` # restituisce le righe che contengono 'Morro' oppure 'morro'
- `$ grep '[^L]inux' <file>` # restituisce tutte le linee che contengono occorrenze che NON hanno un carattere 'L' seguito da 'inux' (potrebbe essere anche 'Sinux', 'tinux', ecc..)
- `$ grep -w linux <file>` # cerca un **parola intera**, case sensitive
- `$ grep '[0-9]' <file>` # cerca la ricorrenza di **un numero da 0 a 9**
- `$ grep -E '[0-9]{4}' <file>` # cerca **4 numeri 0-9 consecutivi**, il parametro '**-E**' interpreta le espressioni regolari estese
- `$ egrep '[0-9]{4}' <file>` # come il precedente, ma deprecato (**egrep = grep -E**)
- `$ fgrep <espressione> <file>` # cerca l'**espressione esatta**, senza interpretare le espressioni regolari. (**fgrep = grep -F**)

Alcuni esempi con sed:

- `$ sed 's/<ricerca>/<sostituzione>/' <file>` # sostituisce la prima ricorrenza (vedi il capitolo su 'sed')
- `$ sed -i 's/<ricerca>/<sostituzione>/' <file>` # modifica direttamente
- `$ sed -i.bak 's/<ricerca>/<sostituzione>/' <file>` # modifica direttamente salvando una copia dell'originale con suffisso `.bak`
- `$ sed '/^linux/d' <file>` # **elimina le righe** che iniziano con la parola 'linux'

ESERCIZIO [1/2]

Il file `/etc/sysctl.conf` contiene molte righe commentate che iniziano con '#'.

Con l'aiuto di **sed**, stampa soltanto le righe NON commentate, quindi il contenuto effettivo del file.

SOLUZIONE

```
cat /etc/sysctl.conf | sed /^#/d
```

ESERCIZIO [2/2]

Ora trova il modo di eliminare anche le righe vuote o "bianche".

SOLUZIONE

```
cat /etc/sysctl.conf | sed /^#/d|sed /^\$/d
```

103.8 - Basic file editing [3]

Introduzione a VI / VIM

VI è un editor minimale ma molto potente, generalmente preinstallato su qualunque distribuzione.

NOTA

VIM = VI Improved.

VIM espande le funzionalità di VI ma nell'uso base le differenze sono poche.

Spesso `vi` è un collegamento a `vim`.

- `$ vi <file>` # apre un file esistente o lo crea

Esistono 3 diverse modalità in cui si può operare in '`vi`':

- '`i`' # modalità inserimento / insert mode
- '`:`' # modalità comando / command mode
- '`ESC`' # modalità movimento / motion mode

Nella modalità *motion mode* la pressione dei tasti esegue delle azioni, iniziamo a vedere come spostarsi con il cursore:

- '`h`' # a sinistra di un carattere
- '`j`' # in basso di una riga
- '`k`' # in alto di una riga
- '`l`' # a destra di un carattere
- '`b`' # all'inizio di ogni parola
- '`w`' # alla fine di ogni parola
- '`e`' # alla fine della parola successiva
- '`(`' # indietro di una frase
- '`)`' # avanti di una frase
- '`{`' # indietro di un paragrafo
- '`}`' # avanti di un paragrafo
- '`$`' # a fine riga
- '`0`' # ad inizio riga
- '`G`' # alla fine del file
- '`gg`' # all'inizio del file

In '`vi`' si può sempre specificare quante volte eseguire un comando premettendo il relativo numero al comando:

- '`5w`' # avanti di 5 parole
- '`5dd`' # elimina le prossime 5 righe
- '`15G`' # alla riga numero 15
- '`:15`' # alla riga 15 ma utilizzando 'command mode'

Copiare e incollare in '`vi`':

- '**yy**' # copia la riga corrente
- '**p**' # incolla dopo il cursore (la riga sotto in questo caso)
- '**P**' # incolla prima del cursore (la riga sopra in questo caso)
- '**10p**' # incolla sotto 10 volte

Vediamo altri comandi da utilizzare in *motion mode*:

- '**dd**' # taglia l'intera riga dove si trova il cursore
- '**yy**' # (yank) copia l'intera riga dove si trova il cursore
- '**p**' # incolla una riga tagliata o copiata in precedenza
- '**u**' # annulla ('undo') in successione comandi o digitazioni
- '**CTRL+r**' # ripete l'azione annullata ('redo')
- '**x**' # elimina il carattere sotto al cursore (come il tasto 'DEL')
- '**X**' # elimina il carattere precedente al cursore (come il tasto 'BACKSPACE')
- '**D**' # elimina dal cursore a fine riga

Cercare del testo con '**vi**':

- '**/**' # cerca la prima occorrenza digitata
- '**n**' # cerca il risultato successivo della ricerca in avanti nel file
- '**N**' # cerca il risultato precedente della ricerca nel file
- '**?**' # cerca a ritroso la prima occorrenza digitata (invertendo l'utilizzo dei comandi 'n' e 'N')
- '**n**' # continua a cerca all'indietro
- '**N**' # continua a cerca in avanti

Alcuni comandi invece da digitare in *command mode*:

- '**:w**' # salva il file
- '**:wq**' # salva il file ed esce da 'vi'
- '**ZZ**' # salva il file SOLO se è stato modificato, ed esce. (equivalente a '**:x**')
- '**:q**' # esce da 'vi'
- '**:q!**' # forza l'uscita da 'vi' anche se il file è stato modificato

Alcuni modi per entrare in *insert mode*:

- '**i**' # entra in insert mode
- '**o**' # crea una riga vuota sotto ed entra in insert mode
- '**O**' # crea una riga sopra ed entra in insert mode
- '**A**' # va a fine riga ed entra in insert mode - (a)ppend
- '**s**' # elimina il carattere dove si trova il cursore ed entra in insert mode
- '**R**' # entra in modalità sostituzione sovrascrivendo i caratteri successivi (come entrare in insert mode e premere 'INS')

Cercare e sostituire testo (molto simile a '**sed**'):

- '**:s/<parola>/<sostituto>**' # sostituisce la prima occorrenza nella riga corrente
- '**:s/<parola>/<sostituto>/g**' # sostituisce tutte le occorrenze nella riga corrente

- '`:%s/<parola>/<sostituto>/`' # sostituisce la prima occorrenza trovata in tutte le righe del file
- '`:%s/<parola>/<sostituto>/g`' # sostituisce tutte le occorrenza trovate nell'intero file
- '`:%s/<parola>/<sostituto>/gc`' # come il precedente ma chiede (c)onferma ad ogni sostituzione
- '`r`' # sostituisce il carattere sotto al cursore con il successivo carattere premuto

Default EDITOR e nano

In Linux esistono altri editor di testo oltre a '`vi`', ad esempio:

- '`nano`'
- '`emacs`'
- '`ed`'

Possiamo impostare l'editor di testo predefinito:

- modificando il link simbolico `/etc/alternatives/editor`
- modificando la variabile `$EDITOR` con il percorso del comando del nostro editor preferito

Questo influenzera l'editor predefinito richiamato da comandi come:

- `visudo` # per modificare il file `sudoers` (vediamo poi)
- `systemctl edit unit` # per modificare un file di unit systemd
- `git commit` # per la scrittura di un messaggio di commit in git
- ...

Appendice: Riassunto comandi più comuni

alias e type

- `$ type <comando>` # distingue il comando tra alias, shell builtin, o percorso dell'eseguibile
- `$ alias <abbreviazione>='<comando-completo>'` # crea un alias per digitare anziché

shell history

- `$ history` # mostra i comandi digitati in precedenza
- `$ history -c` # pulisce la cronologia dei comandi
- '`C-r`' # inizia la ricerca nella history
- `$!!` # esegue nuovamente l'ultimo comando invocato
- `$!$` # richiama il parametro usato nell'ultimo comando invocato

manuale

- `$ man <comando>` # apre il manuale utente per il comando specificato (il manuale utente non è presente per i comandi shell builtin)
- `$ <comando> --help` # mostra l'help per il programma specificato (solitamente più sintetico del man, presente anche per i builtin)

informazioni sul sistema

- `$ uname -a` # mostra tutte le info sul kernel in esecuzione
- `$ uname -r` # mostra solo il numero di versione del kernel

lavorare con i file di testo

Premessa: l'operatore '`>`' ha sempre il compito di redirigere l'output (`stdout`) sul file indicato:

- `$ split -l 100 file.txt` # divide file.txt in 'n' file da 100 righe ciascuno
- `$ split -b 5m video.mp4` # divide video.mp4 (file binario, non di testo!) in porzioni di 5mb
- `$ cat video* > unito.mp4` # unisce nuovamente tutti i file con prefisso nominale 'video' nel file unito.mp4 (funziona con qualunque formato)
- `$ cut -d ':' -f 1 ab.txt` # imposta come separatore di colonna il carattere ':' e stampa la prima colonna
- `$ cut -d ':' -f1,7 ab.txt` # come sopra, ma stampa le colonne da 1 a 7
- `$ head -n 10 file.txt` # mostra solo le prime 10 righe di file.txt
- `$ tail -n 10 file.txt` # mostra solo le ultime 10 righe di file.txt
- `$ wc -l file.txt` # conta il numero di righe presenti in file.txt
- `$ wc -c file.txt` # conta il numero di caratteri presenti in file.txt
- `$ less file.txt` # visualizza file.txt poco per volta. usare le frecce per navigare il documento (Up/Down)
- `$ zcat file.txt.gz` # visualizza file.txt.gz (compresso) senza bisogno di decomprimere
- `$ bzcat file.txt.bz2` # come sopra, per i file compressi in formato .bz2
- `$ xzcat file.txt.xz` # come sopra, per i file compressi in formato .xz

gestione dei file

- `$ pwd` # mostra il percorso della cartella in cui ci troviamo (directory locale)
- `$ ls` # mostra il contenuto della cartella in cui ci troviamo (directory locale)
- `$ cd Scrivania` # ci spostiamo con il terminale nella cartella chiamata 'Scrivania' (deve esistere, con quel nome, nella directory locale)
- `$ du -sh folder` # restituisce la dimensione della cartella 'folder', comprensiva di tutti i file in essa contenuti
- `$ md5sum file` # calcola la somma md5 per 'file', può essere utile se confrontato con una copia del file sicuramente integra
- `$ cp file.txt new.txt` # crea una copia di 'file.txt' chiamata 'new.txt'
- `$ cp -r cart1 cart2` # crea una copia della cartella 'cart1' (e ricorsivamente il suo contenuto) chiamata 'cart2'

- `$ mv old new` # *rinomina il file 'old' come 'new'*
- `$ mv old Desktop/new` # *sposta il file 'old' sul 'Desktop', con il nome cambiato in 'new'*
- `$ rm file.txt` # *elimina 'file.txt'*
- `$ rm -r folder` # *elimina la cartella 'folder' ed il suo contenuto (utilizzare anche il parametro '-f' per rimuovere forzatamente)*
- `$ mkdir folder` # *crea la cartella 'folder'*
- `$ mkdir -p folder/ciao` # *crea la cartella 'folder' e al suo interno la sottocartella 'ciao' con un unico comando*
- `$ gzip file` # *genera un file compresso chiamato 'file.gz' ELIMINANDO il file originale*
- `$ gunzip file.gz` # *estrae 'file.gz' ELIMINANDO la sua versione compressa con estensione .gz*
- `$ tar -czf var.tar.gz var` # *comprime la cartella 'var' in un archivio chiamato 'var.tar.gz'*
- `$ tar -xzf var.tar.gz` # *estrae nella cartella locale il contenuto dell'archivio 'var.tar.gz'*
- `$ dd if=/file of=new` # *copia bit a bit il contenuto di 'file' in 'new', spesso usato per clonare supporti di memorizzazione di massa*
- `$ find / -iname "ciao"` # *cerca su tutto il filesystem i file chiamati 'ciao' senza distinguere tra maiuscole e minuscole ('-iname')*

streams, redireciton e pipes

- `$ ls > out.txt` # *redirige l'output di 'ls' nel file 'out.txt'*
- `$ ls 2> err.txt` # *redirige eventuali errori di 'ls' nel file 'err.txt'*
- `$ ls &> out.txt` # *redirige errori e output di 'ls' nel file 'out.txt'*
- `$ cat < in.txt` # *redirige l'input di 'cat' per leggere 'in.txt' (ma non è necessario usare l'operatore di redirezione con 'cat')*
- `$ cat in.txt` # *stesso effetto del comando precedente*
- `$ prog1 | prog2` # *redirige l'output del programma 'prog1' sull'input di 'prog2'*

gestione processi e risorse

- `'C-z'` # *manda un processo in background (ed in pausa)*
- `$ jobs` # *mostra i processi in background*
- `$ fg <n>` # *riporta in foreground (in primo piano) il processo*
- `$ ps aux` # *mostra tutti i processi in esecuzione sulla macchina*
- `$ top` # *mostra in maniera interattiva tutti i processi in esecuzione, compreso utilizzo delle risorse (stile task manager)*
- `$ htop` # *versione più evoluta di 'top'*
- `$ free -m` # *mostra l'uso di memoria RAM*
- `$ df -h` # *mostra l'uso del disco (spazio libero per ogni partizione)*
- `$ iotop` # *(extra) mostra l'uso del disco (lettura/scritture correnti, e da quali processi)*
- `$ nohup <cmd> &` # *esegue il comando in modo che non venga interrotto quando chiudiamo la shell o ci scolleghiamo*

- `$ kill [-S] <pid>` # uccide il processo con un dato (il parametro '-S' specifica il codice di terminazione come, ad esempio, -15 o -9)
- `$ killall <nome>` # uccide tutti i processi con un certo nome

104 - Dispositivi, Filesystem Linux, Filesystem Hierarchy Standard

104.1 - Create partitions and filesystems [2]

MBR vs GPT

MBR (Master Boot Record), è stato introdotto da IBM nel 1983 ed è caratterizzato dal fatto che i primi 512 bytes del disco contengono il bootloader. Pertanto il BIOS non deve fare altro che leggerli per dare inizio all'avvio della macchina.

MBR ha diverse limitazioni:

- funziona solo con dischi di max 2TB;
- supporta al massimo 4 partizioni primarie.

GPT (GUID Partition Table) è stato progettato da Intel a fine anni 90 ed ha iniziato a vedere vasta adozione intorno al 2010.

Alcune caratteristiche e differenze con MBR:

- Supporta fino a 128 partizioni primarie;
- Ogni partizione ha un identificatore univoco (**GUID** - Global Unique IDentifier);
- La tabella delle partizioni è salvata in più settori del disco => migliore resistenza alle corruzioni del disco;
- Tramite i CRC (Cyclic Redundancy Check), è possibile verificare l'integrità della partition table e tentare la riparazione.

Per formattare un disco usando MBR o GPT posso usare il comando '`parted`':

- `# parted /dev/sdb`
- `(parted) mklabel`

FAT32 vs exFAT ; Btrfs

FAT32 (32 Bit File Allocation Table) è stato per lungo tempo il filesystem più diffuso perché compatibile trasversalmente con Windows, MacOS e Linux, ha però forti limitazioni quali:

- Supporto di una dimensione massima per un file di 4GB;
- È un filesystem molto soggetto a frammentazione.

exFAT, introdotto più recentemente, risolve queste limitazioni mantenendo la compatibilità multipiattaforma:

- Dimensione massima per un file pari a 16EB (1 milione di TB);
- Migliorate le prestazioni in termini di frammentazione.

Per usare il filesystem exFAT dovrò installare:

- `# apt install exfat-fuse # per montare e leggere partizioni exFAT`
- `# apt install exfat-utils # per creare partizioni exFAT`
- `# mkfs.exfat <partizione>`

Btrfs (B-Tree FileSystem) è un filesystem avanzato (ancora in fase di sviluppo) con diverse caratteristiche interessanti:

- supporto nativo per dispositivi multipli (**logical volume management**);
- **compressione** nativa zlib;
- **sotto-volumi** con diverse radici del filesystem;
- supporto nativo agli **snapshot**;
- **conversione in-place** (sul posto) da filesystem ext2/3/4.

Soprattutto l'ultima caratteristica permette di passare a Btrfs senza dover formattare il disco se questo è formattato con ext2, ext3 o ext4.

104.2 - Maintain the integrity of filesystems [2]

In linux, i filesystem hanno un contatore che viene incrementato ad ogni montaggio. In questo modo è possibile programmare un check del filesystem ogni N montaggi con **tune2fs**:

- `# tune2fs -l <partizione>` # genera un po' di informazioni tra cui 'Mount count', 'Maximum mount count' (-1 = mai) e numero di blocchi riservati all'utente root.
- `# tune2fs -c 15 <partizione>` # imposta il numero di montaggi dopo cui effettuare il check: '-1' significa mai, '1' significa ogni volta che viene montato
- `# tune2fs -i 15` # imposta il check al boot ogni 15 giorni, a prescindere dal numero di boot

Nel file `/etc/fstab` sono elencate le partizioni di cui facciamo il `mount` in fase di boot.

Il valore indicato alla sesta colonna (l'ultima) può essere:

- '`0`': significa che non è previsto fare il check della partizione
- '`1, 2, ...`': Check abilitato, il numero stabilisce l'ordine di check rispetto alle altre partizioni.

NOTA

Vedremo in seguito il significato di ogni colonna del file `fstab`.

Il controllo può anche essere lanciato **manualmente** tramite l'utility '`fsck`', che sul sistema è presente con vari comandi a seconda del tipo di *filesystem* che si vuole controllare:

- `# e2fsck # per le famiglie 'ext2', 'ext3', 'ext4'`
- `# dosfsck # per le famiglie 'fat' (è un link simbolico a 'fsck.fat')`

- # `reiserfsck` # per i filesystem di tipo 'reiser'
- # `xfs_repair` # per filesystem di tipo 'XFS' (necessita del pacchetto 'xfsprogs')

Il filesystem **XFS** include anche una serie di altre utility:

- # `xfs_fsr` # esegue l'equivalente di un defrag
- # `xfs_db` # sta per "xfs debug" e può essere utilizzato per analizzare un filesystem XFS.

NOTA

Non è necessario ricordare tutti questi comandi: è possibile utilizzare `fsck`.

`<filesystem>` allo stesso scopo.

Controlla i filesystem supportati con `fsck .<TAB><TAB>`.

Non è possibile fare il check delle partizioni montate. Se possibile, smonta la partizione prima di fare il check.

Alternativamente è possibile ri-montare una partizione in modalità read-only (RO):

- # `mount -o remount,ro <partizione>` # rimonta la partizione in sola lettura

E forzare il check:

- # `e2fsck <partizione> -f`

Ma il check di partizioni montate è una operazione fortemente sconsigliata.

Vediamo un paio di utilizzi dell'utility '`fsck`':

- # `fsck -A` # fa il controllo di tutte le unità che devono essere controllate in base alle impostazioni che sono scritte nel file '/etc/fstab'
- # `fsck -a` # esegue la riparazione in automatico senza chiedere conferma

ESERCIZIO

1. Fà in modo che, al prossimo riavvio, il filesystem radice (/) venga controllato.
2. Riavvia e osserva il risultato
3. Accertati che il filesystem radice non venga controllato ad ogni avvio da ora in poi.

Blocchi riservati

I filesystem ext3 ed ext4 riservano sempre un certo numero di blocchi per l'utente root.

In questo modo, se il disco dovesse arrivare a riempirsi completamente, sarà comunque possibile, per l'utente root, collegarsi ed effettuare operazioni di manutenzione al fine di "sbloccare" il sistema.

Per impostazione predefinita, i blocchi riservati all'utente root corrisponde al 5% dei blocchi totali disponibili, ma questo valore può essere impostato arbitrariamente con `tune2fs`:

- # `tune2fs -r` # imposta il **numero di blocchi riservati** per l'utente root

- # `tune2fs -m` # imposta la **percentuale** di blocchi riservati per l'utente root

debugfs (rimosso in 101-500)

'`debugfs`' è una shell interattiva per analizzare i filesystem ext2, ext3 ed ext4 e che, tra le altre cose, permette di mostrare l'*inode* a cui corrisponde un file e **anche gli inode dei file cancellati** di recente, per cercare di recuperarli. (questa procedura di *undelete* si è complicata dalla versione ext3 in poi):

- # `debugfs` # lancia l'utility in modalità interattiva
- `debugfs: ?` # visualizza l'elenco dei comandi disponibili
- `debugfs: open <partizione>` # 'apre' una partizione
- `debugfs: pwd` # verifica in quale directory siamo
- `debugfs: cd <cartella>` # cambia directory
- `debugfs: ls` # elenca files e directories con i relativi 'inode'
- `debugfs: ls -d` # elenca anche i files e directries (questi ultimi con inode tra <>)
- `debugfs: quit` # esce dalla shell interattiva di debugfs

dumpe2fs (rimosso in 101-500)

'`dumpe2fs`' è un altro comando che fornisce informazioni sulle partizioni ext2, ext3 ed ext4:

- # `dumpe2fs -h <partizione>` # fornisce informazioni analoghe a 'tune2fs -l'

104.3 - Control mounting and unmounting of filesystems [3]

Montaggio e punto di mount

Da ora in avanti useremo il generico termine "*block device*" per riferirci ai dispositivi di archiviazione di massa, indipendentemente dal fatto che si tratti di memorie flash, HDD meccanici, dischi SSD o quant'altro.

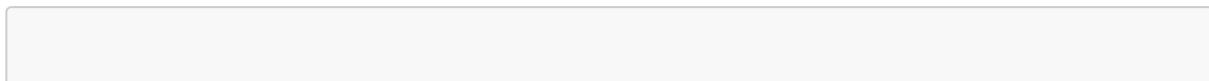
Generalmente, qualsiasi block device è configurato per avere **una o più partizioni**. Su ciascuna partizione può essere **creato un filesystem** differente per conservare file di varia natura.

L'operazione di *rendere accessibili i dati sul filesystem* per la lettura e/o la scrittura è detta **montaggio**. Con questa operazione si vede anche a definire *il percorso a cui il filesystem montato sarà accessibile*, anche detto **punto di mount** o punto di montaggio.

Per **elencare i block device** con le relative partizioni possiamo usare `fdisk` oppure `parted` come segue:

- # `fdisk -l` # mostra i block device fisicamente collegati
- # `parted -l` # idem, ma più accurato e completo.

Ad esempio:



```
[morro@t480s ~]$ sudo parted -l
Modello: Samsung SSD 980 500GB (nvme)
Disco /dev/nvme0n1: 500GB
Dimensione del settore (logica/fisica): 512B/512B
Tabella delle partizioni: gpt
Flag del disco:

Numero Inizio Fine Dimensione File system Nome
Flag
 1      1049kB 630MB   629MB     fat32       EFI System
Partition avvio, esp
 2      630MB  1704MB  1074MB    ext4
...
...
```

Fornisce un sommario completo sullo schema di partizionamento e filesystem su ciascuna partizione per ogni block device collegato.

Per **elencare tutti i filesystem** attualmente montati possiamo usare:

- `$ mount # (senza opzioni) mostra tutti i filesystem montati, compresi i filesystem di rete e virtuali.`
- `$ cat /proc/self/mounts # equivalente. Elenca tutti i filesystem montati.`

Per **montare il filesystem** di una partizione specificata:

- `# mount <partizione> </punto/di/mount> # monta il filesystem su "partizione" al percorso "/punto/di/mount"`

Ad esempio:

- `# mount /dev/nvme0n1p1 /boot/efi`

Monta il filesystem `fat32` della partizione `1` del disco `/dev/nvme0n1` rendendola accessibile al percorso `/boot/efi`.

NOTA

Il punto di mount indicato al comando `mount` **deve essere una cartella vuota**:

- Se la cartella indicata come punto di mount **non è vuota**, il suo contenuto sarà mascherato dal contenuto del filesystem montato e non risulterà accessibile fino allo smontaggio del suddetto filesystem
- Se la cartella indicata come punto di mount **non esiste**, riceveremo un errore "mount point not found".

Come anticipato, montando un filesystem su una cartella non vuota, i contenuti della suddetta cartella saranno mascherati dal nuovo filesystem. Per rendere nuovamente accessibili i contenuti della cartella senza smontare il filesystem che la maschera, è possibile effettuare un **bind mount**:

```
# creo una cartella con 10 file
[morro@t480s Scaricati]$ touch cartella/file{1..10}
[morro@t480s Scaricati]$ ls cartella/
file1  file10  file2  file3  file4  file5  file6  file7  file8
file9

# monto una immagine iso usando questa "cartella" come punto di
mount
[morro@t480s Scaricati]$ sudo mount -o loop Zorin-OS-16.3-Core-64-
bit.iso cartella/
mount: /home/morro/Scaricati/cartella: WARNING: source write-
protected, mounted read-only.

# il contenuto originale di "cartella" non è più visibile!
[morro@t480s Scaricati]$ ls cartella/
boot  casper  dists  efi  isolinux  md5sum.txt  pool
README.diskdefines

# creo "cartella2" e monto al suo interno il contenuto di
"cartella"
[morro@t480s Scaricati]$ mkdir cartella2
[morro@t480s Scaricati]$ sudo mount --bind cartella cartella2

# ora anche "cartella2" è usata come punto di mount per il
filesystem montato:
[morro@t480s Scaricati]$ ls cartella2/
boot  casper  dists  efi  isolinux  md5sum.txt  pool
README.diskdefines

# smonto "cartella" per recuperare accesso ai file in essa
contenuti
[morro@t480s Scaricati]$ sudo umount cartella
[morro@t480s Scaricati]$ ls cartella/
file1  file10  file2  file3  file4  file5  file6  file7  file8
file9

# il filesystem rimane montato su "cartella2":
[morro@t480s Scaricati]$ ls cartella2
boot  casper  dists  efi  isolinux  md5sum.txt  pool
README.diskdefines
```

PRO TIP

Il binding di mount è anche usato nelle operazioni di `chroot` per rendere disponibili al sistema "montato" i punti di mount dei filesystem virtuali come `/dev/`, `/run/`, ecc...

Il comando `mount` supporta numerose altre opzioni, ad esempio:

- `mount -a` # monta tutti i filesystem definiti in `/etc/fstab`.

- `mount -t` # permette di specificare il tipo di filesystem manualmente, in caso di problemi

Consulta il `man mount` per maggiori informazioni.

Per **smontare un filesystem** usiamo il comando `umount` indicando il punto di montaggio utilizzato in precedenza:

- `# umount <punto-di-mount>` # smonta la partizione montata al punto di mount indicato.

Il punto di mount può essere qualunque percorso, ma per il montaggio di unità generiche (senza uno scopo preciso) è generalmente consigliato fare uso di uno dei seguenti percorsi

- `/mnt/<nome-disco>` # NOTA: è una cartella da creare appositamente!
- `/media/<nome-disco>` # NOTA: è una cartella da creare appositamente!

Alcuni desktop environment montano automaticamente i block device collegati in percorsi come:

- `/run/media/$USER/<disk-uuid>`

NOTA

Il montaggio con `mount` **non è persistente**, ovvero i filesystem montati in questo modo non rimangono montati ai riavvii successivi. Per rendere i mount persistenti possono essere usati `fstab` e le `systemd mount units` trattate di seguito.

ESERCIZIO

1. Assicurati di avere un secondo disco collegato alla macchina virtuale o una chiavetta USB.
2. Crea un punto di mount per il disco secondario `/home/$USER/disco2`
3. Assicurati che sul disco sia presente almeno una partizione `ext4` e montala nel mountpoint indicato al punto precedente.
4. Controlla che la partizione sia stata montata correttamente con il comando `mount`.

Il file fstab

Per **rendere persistente il montaggio di un filesystem** ad ogni reboot, dobbiamo creare una regola di mount nel file `/etc/fstab`.

Il file `/etc/fstab` è suddiviso in 6 colonne di informazioni:

- `<device>` # identifica la partizione da montare (ES: `/dev/nvme0n1p1` oppure il relativo identificatore univoco `UUID`)
- `<mount point>` # specifica il punto di mount
- `<type>` # precisa la tipologia di filesystem contenuta nella partizione
- `<option>` # elenca le varie opzioni di montaggio
- `<dump>` # viene letto dal comando 'dump', se pari a '0' non necessita del suo intervento
- `<pass>` # viene letto dal comando 'fsck' e, se pari a '0' non necessita del suo intervento

Ad esempio:

```
[morro@t480s ~]$ cat /etc/fstab
...
UUID=E78B-FD72      /boot/efi          vfat
umask=0077, shortname=winnt 0 2
...
```

Assicura che il filesystem **fat32** (anche detto **vfat**) sulla partizione **/dev/nvme0n1p1** sia montato automaticamente ad ogni avvio nel percorso **/boot/efi**.

NOTA

Nel primo campo di **fstab**, non abbiamo usato il "nome kernel" del dispositivo e partizione (**/dev/nvme0n1p1**) per riferirci ad esso, ma il suo **UUID**: un identificatore univoco che **non cambia** in base al numero e all'ordine dei block device collegati.

Per **ottenere lo UUID** di una partizione è possibile usare il comando **blkid**:

- # **blkid /dev/nvme0n1p1** # mostra lo UUID della partizione **/dev/nvme0n1p1**

È sempre consigliato utilizzare lo **UUID** della partizione indicata al posto del "nome kernel" all'interno di **fstab**.

PRO TIP

L'inserimento di una regola errata in **/etc/fstab** può causare un errore di montaggio al prossimo reboot, **impedendo il corretto avvio del sistema**.

Si consiglia sempre di **verificare** che tutte le regole di mount funzionino correttamente dopo aver modificato il file **fstab** *forzando la loro esecuzione immediata* con il comando **mount -a**.

PRO TIP

Tutti i block device indicati in **fstab** devono essere presenti e collegati in fase di boot, o si otterrà un errore che impedisce il corretto avvio del sistema. Se questo non può essere garantito (ES: unità flash removibile) si consiglia di aggiungere l'opzione **nofail** nella colonna opzioni di **fstab**.

ESERCIZIO

1. Rendi *permanente* il montaggio del filesystem dell'esercizio precedentemente con una regola in **fstab**.
2. Verifica la correttezza
3. Riavvia per controllare che il montaggio avvenga in automatico.

lsblk e blkid

lsblk mostra i block devices (fisici e virtuali) e relativi mount point:

```
[morro@t480s ~]$ lsblk
NAME                           MAJ:MIN RM  SIZE RO
TYPE   MOUNTPOINTS
sda                            8:0    1   0B  0
disk
zram0                          252:0   0   8G  0
disk   [SWAP]
nvme0n1                         259:0   0 465,8G  0
disk
└─nvme0n1p1                     259:1   0 600M  0
part   /boot/efi
└─nvme0n1p2                     259:2   0   1G  0
part   /boot
...
...
```

blkid mostra UUID e PARTUUID della partizione, disco o dispositivo indicato:

- # **blkid <device>**

Vediamo la **differenza tra UUID e PARTUUID**:

- '**UUID**' # ID della partizione assegnato da **Linux**
- '**PARTUUID**' # ID della partizione assegnato da **GPT**

Nel file **/etc/fstab** posso indicare indifferentemente l'uno o l'altro.

systemd mount units

Un metodo alternativo per montare dischi e partizioni all'avvio è tramite la creazione di una '**mount unit**'.

Se ad esempio volessi montare una partizione nel percorso **/mnt/part4**, dovrò creare un file di configurazione nel percorso **/etc/systemd/system/mnt-part4.mount** dove per convenzione '**-**' viene usato al posto di **/**:

```
[Unit]
Description=Monta la partizione 4

[Mount]
What=/dev/disk/by-uuid/UUID-della-partizione
Where=/mnt/part4
Type=ext4
Options=defaults

[Install]
WantedBy=local-fs.target
```

Successivamente:

- `# systemctl daemon-reload` # carica la nuova 'unit' (va eseguito SEMPRE quando si modifica o crea una nuova unit)
- `# systemctl start mnt-part4.mount` # monta la partizione
- `# systemctl status mnt-part4.mount` # verifica che l'unità è stata caricata e di conseguenza la partizione montata/
- `# systemctl enable mnt-part4` # imposta il montaggio della partizione al boot

NOTA

A prescindere dal metodo, quando viene montato un filesystem (anche manualmente) systemd genera al volo una "mount unit" per monitorare lo stato del mount.

Per elencare tutte le mount unit caricate da systemd (sia quelle definite staticamente che quelle generate automaticamente) possiamo usare il comando:

```
systemctl list-units --type=mount

UNIT                                     LOAD   ACTIVE SUB      DESCRIPTION
-.mount                                  loaded active mounted Root Mount
boot-efi.mount                            loaded active mounted /boot/efi
boot.mount                                loaded active mounted /boot
dev-hugepages.mount                       loaded active mounted Huge Pages
File System
  dev-mqueue.mount                         loaded active mounted POSIX
Message Queue File System
  home.mount                               loaded active mounted /home
  run-user-1000-doc.mount                  loaded active mounted
/run/user/1000/doc
  run-user-1000-gvfs.mount                loaded active mounted
/run/user/1000/gvfs
  run-user-1000.mount                     loaded active mounted
/run/user/1000
    sys-fs-fuse-connections.mount        loaded active mounted FUSE
Control File System
  sys-kernel-config.mount                loaded active mounted Kernel
Configuration File System
  sys-kernel-debug.mount                 loaded active mounted Kernel
Debug File System
  sys-kernel-tracing.mount              loaded active mounted Kernel
Trace File System
  tmp-.mount_kDrivevPYDMi.mount         loaded active mounted
/tmp/.mount_kDrivevPYDMi
  tmp.mount                               loaded active mounted Temporary
Directory /tmp
  var-lib-nfs-rpc_pipefs.mount          loaded active mounted RPC Pipe
File System

LOAD  = Reflects whether the unit definition was properly loaded.
```

ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
 SUB = The low-level unit activation state, values depend on unit type.

ESERCIZIO

1. Rimuovi la regola di **fstab** creata nell'esercizio precedente e crea una *systemd mount unit* allo stesso scopo
2. Riavvia e verifica il corretto funzionamento.

104.5 - Manage file permissions and ownership [3]

leggere i permessi

Il sistema di permessi in linux è piuttosto elementare.

A ciascun file (o cartella) è assegnato:

- un **utente proprietario (user)** - per garantire al proprietario uno specifico set dei permessi.
- un **gruppo proprietario (group)** - per garantire agli utenti che appartengono al gruppo un altro set di permessi.
- e per **tutti gli altri utenti (others)** - può essere definito un livello di privilegio comune.

Le informazioni sui permessi e sul proprietario di un file (o cartella) possono essere ottenute con il comando **ls -l**:

```
morro@air $ ls -lh LPIC1-101.md
-rwxr-xr-x 1 morro    staff   195K 27 Set 17:13 LPIC1-101.md
      ^        ^        ^
      |        |        |
permessi    |        gruppo          nome
            |        proprietario     del file
            utente
            proprietario
```

In questo caso il file:

- appartiene all'**utente morro** (che l'ha creato) e
- appartiene al **gruppo staff** (gruppo primario di "morro").

La colonna dei permessi sul file

```
-rwxr-xr-x
```

È in realtà divisa in 1 + 3 triplette

```
- rwx    r-x    r-x
      ^     ^     ^
      u     g     o
```

che si riferiscono a:

- '**u**' # (*u*ser - utente proprietario del file o cartella)
- '**g**' # (*g*roup - gruppo di appartenenza del file o cartella)
- '**o**' # (*o*thers - tutti gli altri utenti che non sono i proprietario e non fanno parte del gruppo a cui appartiene il file o la cartella)

Ciascuna tripletta è usata per rappresentare 3 permessi:

- '**r**' # **lettura**
- '**w**' # **scrittura**
- '**x**' # **esecuzione**

Quindi ad esempio, la tripletta:

- **rwx** # indica permesso di **lettura, scrittura ed esecuzione**
- **rw-** # indica **lettura e scrittura** ma NON esecuzione
- **r--** # indica **lettura** ma NON scrittura e NON esecuzione

E così via.

La prima tripletta è riferita all'utente (u), la seconda tripletta al gruppo (g) e la terza a tutti gli altri (o).

Di conseguenza, per la colonna di permessi

```
- rwxr-xr-x
```

Avremo che:

- L'**utente proprietario** può leggere, scrivere ed eseguire (**rwx**)
- I membri del **gruppo proprietario** possono leggere ed eseguire (**r-x**)
- Tutti **gli altri** possono leggere ed eseguire (**r-x**) parimenti.

Il primo carattere nella colonna dei permessi di **ls**:

```
- rwx r-x r-x
^  
questo
```

Rappresenta il **tipo di file**:

- **-** simboleggia un file regolare
- **d** simboleggia una directory
- **l** simboleggia un link simbolico
- **s** simboleggia un socket
- ...

Leggi il **man ls** per una enumerazione esaustiva.

NOTA

Impostare il permesso di **esecuzione su una cartella** significa permettere **l'attraversamento** di quella cartella.

Cambiare i permessi - chmod

I permessi su file e directory possono essere cambiati tramite comando '**chmod**'.

Il comando **chmod** accetta **due tipi di notazioni**:

- Notazione **simbolica**
- Notazione **ottale**

NELLA NOTAZIONE SIMBOLICA usiamo:

- **u, g, o** per riferirci ai soggetti
- **r, w, x** per riferirci ai permessi

Ad esempio:

- **\$ chmod u+x <file>**

aggiunge (+) il permesso di esecuzione (x) all'utente proprietario (u), mentre

- **\$ chmod ugo-x <file>**

rimuove (-) il permesso di esecuzione (x) al proprietario (u), agli utenti che fanno parte del gruppo (g) ed a tutti gli altri (o)

- **\$ chmod u=rw <file>**

imposta esattamente (=) i permessi di lettura (r) e scrittura (w) per l'utente proprietario (u), **rimuovendo** il permesso di esecuzione qualora fosse presente.

PRO TIP

Quando si utilizza la notazione simbolica, per riferirsi a **tutte e 3** le triplette (ugo) è possibile omettere la parte a sinistra del permesso: **chmod ugo+x** equivale a **chmod +x**.

NOTA

Tramite **chmod** è possibile applicare i permessi *in cascata* su tutti i file e directory figlie utilizzando l'opzione di ricorsività (**-R**) sulla directory padre.

LA NOTAZIONE OTTALE è più sintetica e può essere usata per impostare i permessi di utente, gruppo e others **simultaneamente**.

Al posto delle lettere **r w x** usiamo i numeri **4 2 1** per simboleggiare lettura, scrittura ed esecuzione:

- '**r**' = '**4**'
- '**w**' = '**2**'
- '**x**' = '**1**'

Il vantaggio dei numeri è che **possono essere sommati**, perciò:

- '**rw**' = '**6**' $r+w = 4+2$
- '**rx**' = '**5**' $r+x = 4+1$
- '**rwx**' = '**7**' $r+w+x = 4+2+1$

E via dicendo.

In questo modo, **riassumiamo una tripletta **rwx** in una sola cifra** e possiamo rappresentare l'intera colonna dei permessi con soltanto 3 cifre:

- **\$ chmod 775 <file>** # corrisponde a '**rwxrwxrw-**'.
- **\$ chmod 644 <file>** # corrisponde a '**rw-r--r--**'.
- **\$ chmod 444 <file>** # corrisponde a '**r--r--r--**'.

Per chiarire:

r w -	r - -	r - -	\Rightarrow	rw-r--r--
4 2	4	4	\Rightarrow	644

oppure

r w x	r w -	r - -	\Rightarrow	rwxrw-r--
4 2 1	4 2	4	\Rightarrow	764

E via dicendo.

ESERCIZIO

1. Crea una cartella chiamata **dir1** con all'interno due file: **file1** e **file2**.
2. Annota i permessi di default su tutti gli elementi appena creati.

3. Rimuovi il permesso di esecuzione per tutti gli utenti su **dir1**. **Cosa è cambiato?**

sticky-bit

Normalmente, gli utenti che hanno il **permesso di scrittura** ('**w**') su un file possono anche **rinominare ed eliminare** quel file, *anche se non ne sono proprietari* (ad esempio '**g**' oppure '**o**').

Per conferire la possibilità di eliminare e rinominare i file **solo al proprietario**, lasciando il permesso di scrittura per tutti gli altri, è possibile **applicare lo "sticky-bit"** alla **cartella padre**:

- \$ chmod +t <cartella>

Trasforma la colonna dei permessi da così:

```
rw-r-xr-x
```

a così

```
rw-r-xr-t  
^sticky bit
```

Nota che lo "sticky bit" copre la posizione del bit di esecuzione ('x') per il gruppo others (l'ultima tripletta a dx). Per questa ragione, verrà rappresentato come:

- '**t**' minuscola se **others** ha il permesso di esecuzione
- '**T**' maiuscola se **others** NON ha il permesso di esecuzione

Quindi, se lo scenario iniziale precedente fosse stato:

```
rw-r-xr--  
^no esecuzione per others!
```

Avremmo avuto

```
rw-r-xr-T  
^sticky bit maiuscolo!
```

Anche in **NOTAZIONE OTTALE** è possibile impostare lo *sticky-bit* aggiungendo '**1**' a sinistra, ad esempio:

- \$ chmod 1777 <cartella> # corrisponde a '**rwxrwxrwt**'.

- `$ chmod 1770 <cartella> # corrisponde a 'rwxrwx--T'.`
- `$ chmod 0777 <cartella> # corrisponde a 'rwxrwxrwx'. # NO sticky-bit.`

Con lo sticky-bit, i files presenti all'interno della cartella potranno essere eliminati solamente dal proprietario del file (o dal proprietario della cartella, come per i file).

La cartella di sistema `/tmp/` ha lo *sticky-bit* e ne si capisce il senso: chiunque può scrivere nella cartella ma solo il proprietario del file creato può eliminarlo.

ESERCIZIO

1. Crea una cartella `/var/dir2` con all'interno due file: `f1` ed `f2`.
2. Applica a `/var/dir2` e tutti i suoi elementi i permessi di lettura, scrittura ed esecuzione per tutti gli utenti.
3. Applica lo sticky bit a `/var/dir2`.
4. Apri una shell come un utente differente (es: "demo") e prova ad eliminare il file `/var/dir2/f1`

SUGGERIMENTO

puoi usare il comando `adduser` per aggiungere un nuovo utente. Consulta il `man adduser` per maggiori informazioni.

set-UID e set-GID

Consentono di eseguire un file con gli stessi privilegi del proprietario (*set-UID*) o del gruppo (*set-GID*) di appartenenza del file stesso.

Questo è generalmente molto utile per permettere ad un utente non privilegiato di eseguire alcune operazioni privilegiate.

Per questioni di sicurezza si può applicare soltanto agli eseguibili compilati e non agli script.

Il *set-UID* e *set-GID* vengono identificati con il carattere '`s`' che si sostituisce al carattere '`x`' rispettivamente ai permessi del proprietario o del gruppo.

Ad esempio, questo:

```
rwxr-x---
```

diventa

```
rwsr-x---
```

^

SUID

per indicare il bit di set-UID,

```
rwxr-s---
  ^
  SGID
```

per indicare il bit di set-GID, oppure

```
rwsr-s---
  ^
  SUID SGID
```

per indicare la presenza di entrambi.

sUID e sGID possono essere impostati con `chmod` in **modalità simbolica**:

- '\$ `chmod u+s`' # sUID
- '\$ `chmod g+s`' # sGID

Oppure in **modalità ottale** in testa alla tripletta come per lo *sticky-bit*:

- '4' # sUID
- '2' # sGID
- '1' # sticky-bit

A esempio:

- \$ `chmod 4775 <file>` # solo set-UID
- \$ `chmod 6644 <file>` # sia set-UID che set-GID
- \$ `chmod 7600 <file>` # contemporaneamente set-UID, set-GID e sticky-bit

ESERCIZIO [1/2]

Cerca ed elenca tutti i file setuid e setgid sul tuo sistema.

SUGGERIMENTO: usa il comando `find`.

ESERCIZIO [2/2]

Esegui il primo degli eseguibili suid trovato e osserva con che utente esegue il processo.

SUGGERIMENTO: usa il comando `ps aux | grep <comando>` per trovare il processo.

Permessi speciali ed effetti speciali

I permessi speciali cambiano di significato a seconda di dove vengono applicati.

Perm

Effetto sui file

Effetto sulle cartelle

Perm	Effetto sui file	Effetto sulle cartelle
suid	Esegue con i privilegi dell'utente proprietario	-
sgid	Esegue con i privilegi del gruppo proprietario	Nuovi file -> gruppo proprietario = a quello della cartella
sticky	-	Solo il proprietario di un file può rinominarlo o modificarlo

Presta molta attenzione a questa tabella per evitare spiacevoli incidenti di sicurezza.

umask

umask è un parametro utile per alterare i permessi di default su file e cartelle alla loro creazione.

Per impostazione predefinita, i permessi applicati sarebbero:

- **0666** per i file, e
- **0777** per le cartelle

Il comando '**umask**' restituisce la maschera che viene applicata in maniera *sottrattiva* ai permessi di default sopracitati. I permessi di default risultanti saranno quindi

- **0666** - **umask** per i file, e
- **0777** - **umask** per le cartelle.

Ad esempio, se il valore di '**umask**' è pari a '**0022**', i nuovi file saranno creati con permessi:

- **0644** per i file, e
- **0755** per le cartelle.

Vediamo perché.

Ad esempio, per i file abbiamo permessi di default a 0666, e supponiamo di avere una umask a 0022.

Con una semplice sottrazione in colonna otteniamo:

$$\begin{array}{r}
 0666 \\
 - \\
 0022 \\
 \hline
 0644
 \end{array}$$

idem per le cartelle (default = 0777) e umask a 0022:

$$\begin{array}{r}
 0777 \\
 - \\
 0022 \\
 \hline
 \end{array}$$

0755

Posso **cambiare il valore di umask** con l'analogo comando:

- `$ umask <nuova-maschera-ottale>`

Da questo momento in poi, tutti i nuovi file e cartelle saranno creati utilizzando la nuova maschera.

NOTA

Questa modifica **interessa soltanto l'utente** che ha eseguito `umask` e **non è persistente** ai reboot successivi.

ESERCIZIO

Modifica il valore di umask a `1444`. E crea un nuovo file. Con quali permessi di default è stato creato?

chown e **chgrp**

Quando un *utente* crea un *file* (o una cartella):

- Ne diventa automaticamente l'*utente proprietario*
- Il "gruppo proprietario" sarà impostato al "gruppo primario" di cui l'utente fa parte (salvo l'uso di sgid sulla cartella)

Per **cambiare il proprietario** di un file (o cartella) usiamo i comandi `chown` e `chgrp`:

- `# chown morro appunti.txt` # imposta `morro` come **utente proprietario del file appunti.txt**
- `# chown morro:staff appunti.txt` # imposta `morro` come **utente** e `staff` come **gruppo proprietario**
- `$ chgrp staff appunti.txt` # imposta `staff` come **gruppo proprietario di appunti.txt**.

Nota bene:

- per cambiare il gruppo di un file ne si deve essere il proprietario;
- per cambiare il proprietario di un file si deve essere *root*.

Consulta il `man chown` `man chgrp` per maggiori informazioni sull'utilizzo.

ESERCIZIO \

1. Crea una cartella `temp3` all'interno della tua home con all'interno un semplice file di testo `file3`.
2. Cambia utente e gruppo proprietario di `temp3` per combaciare l'utente secondario (es: `demo`).

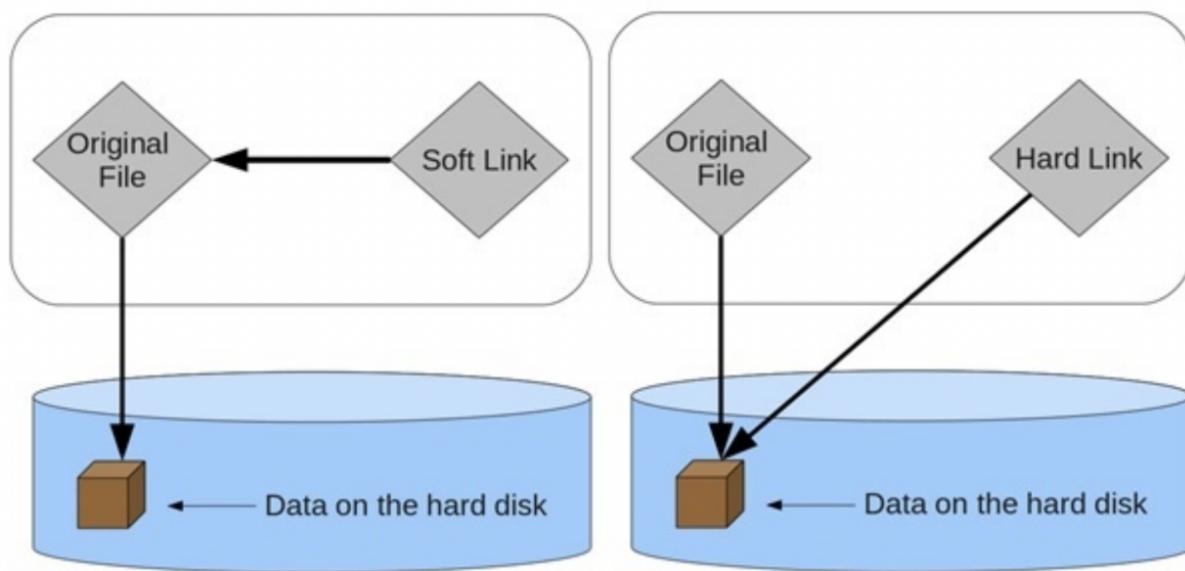
3. Apri una shell come utente secondario e prova ad accedere al file contenuto in **temp3**.
4. Ha funzionato? Perché?

104.6 - Create and change hard and symbolic links [2]

symlink e hardlink in Linux

In linux possiamo avere due tipi di collegamenti: collegamenti **simbolici** (come su Windows) e **hard links**.

- Un **hard link** non è altro che un puntatore diretto ai dati del file. Ogni file sul filesystem ha almeno un "hard link" che punta ad esso.
- Un **collegamento simbolico** è un semplice "collegamento", all'hard link che punta ai dati.



Per creare un **collegamento simbolico** uso **ln -s**:

```
[morro@t480s ~]$ ln -s ciao.png ciao2.png
[morro@t480s ~]$ ls -l ciao*
lrwxrwxrwx. 1 morro morro 8 12 ott 20.20 ciao2.png -> ciao.png
-rw-r--r--. 1 morro morro 481 29 set 15.01 ciao.png
```

Il collegamento creato in questo modo è un file che **punta** al file originale. Se il file originale viene rinominato o spostato, il collegamento si rompe.

Un **hard link**, al contrario, è un tipo di collegamento che **non segue il percorso del file** ma il suo **inode** sul filesystem. In questo caso, se il file originale viene rinominato o spostato, il collegamento continua a funzionare, perché l'inode del file originale non è cambiato!

Per creare un hard link uso **ln** senza opzioni:

```
[morro@t480s ~]$ ln ciao.png ciao3
[morro@t480s ~]$ ls -l ciao*
lrwxrwxrwx. 1 morro morro 8 12 ott 20.20 ciao2.png -> ciao.png
-rw-r--r--. 2 morro morro 481 29 set 15.01 ciao3
-rw-r--r--. 2 morro morro 481 29 set 15.01 ciao.png
```

In questo caso, il collegamento (ciao3) appare come una copia esatta del file originale, tant'è che possiamo anche **eliminare il file originale** e continuare ad accedere agli stessi dati tramite l'hardlink "ciao3".

Non a caso, se utilizziamo l'opzione **-i** di **ls** per mostrare l'**inode** relativo a ciascun file, notiamo che **ciao.png** e **ciao3** **riportano lo stesso inode**, ovvero puntano alla stessa "area" di dati sul filesystem.

```
[morro@t480s ~]$ ls -li ciao*
376054 lrwxrwxrwx. 1 morro morro 8 12 ott 20.20 ciao2.png ->
ciao.png
335448 -rw-r--r--. 2 morro morro 481 29 set 15.01 ciao3
335448 -rw-r--r--. 2 morro morro 481 29 set 15.01 ciao.png
```

Un hard link non è una "copia a peso zero" del file, ma di una "**copia al puntatore del file**": se i dati puntati cambiano, la modifica è osservabile da entrambi i "puntatori" o "hard link".

Nozioni fondamentali:

- '**nome-file**' e '**hardlink**' **puntano ai medesimi dati**, quindi al medesimo *inode*;
- un '**symlink**' contiene semplicemente il *path* per raggiungere '**nome-file**';
- un **hardlink** può riferirsi soltanto a dati **sullo stesso filesystem**;
- un **softlink** può riferirsi a qualsiasi percorso, **anche su filesystem differenti**.
- un '**hardlink**' punta sempre correttamente, perchè punta all'*inode* dei dati.
- un '**symlink**' **punta ad un nome file**. Se il file puntato viene spostato, il **symlink** risulterà *rotto*.
- spostare un '**hard-link**' su un filesystem differente comporta la copia dei dati puntati sul nuovo filesystem. Se non ci sono altri hardlink sul filesystem di origine, il file originale risulterà spostato.

NOTA: il numero che segue la colonna dei permessi nell'output di **ls -l** è il numero di '**hard-links**' che puntano al contenuto del file in oggetto:

```
morro@air $ ls -lh LPIC1-101.md
-rwxr-xr-x 1 morro staff 195K 27 Set 17:13 LPIC1-101.md
^
```

```
|  
num. hardlinks
```

Il comando `ls -i` mostra anche l'*inode* effettivo dei dati a cui punta il file.

```
morro@air $ ls -i LPIC1-101.md  
19441206 LPIC1-101.md  
^  
|  
inode del file
```

ESERCIZIO

1. Crea un file di testo chiamato `f.txt` con contenuto "morrolinux.it"
2. Crea un link simbolico (soft link) a `f.txt` chiamato `l1` nella stessa cartella
3. Crea un hard link a `f.txt` chiamato `h1` nella stessa cartella
4. Sposta `f.txt` su un filesystem differente (ad es nella cartella `/run`)
5. Osserva e commenta il comportamento di `h1` ed `l1`. Cosa è successo?

104.7 - Find system files and place files in the correct location [2]

type, which, whereis, FHS

- `$ type <comando>` # mostra il tipo di oggetto sottostante (comando built-in, eseguibile con percorso, alias, ...);
- `$ which <comando>` # restituisce il primo percorso del comando trovato in `$PATH`
- `$ whereis <comando>` # restituisce il percorso ed il manuale del comando

Filesystem Hierarchy Standard (FHS) è uno standard che definisce le directory principali ed il loro contenuto nel *file-system* dei sistemi operativi Unix-like.

Il documento dello standard nella versione più recente (3.0) può essere trovato sul sito di [Linux Foundation](#).

Di seguito una rapida e non esaustiva rappresentazione dal sito [GeeksForGeeks](#):

