

# UNIX System Administration

Laboratorio di Reti AA 2023/2024  
Ing. Filippo Poltronieri  
[filippo.poltronieri@unife.it](mailto:filippo.poltronieri@unife.it)

# Concetti

- Unix, Linux come sistemi operativi fondamentali
- Versioni custom (small) di Linux sono alla base della maggior parte dei nostri dispositivi casalinghi (router / access-point)
- Processi Demone
- Conoscere i concetti cardine per la gestione di un sistema e l'accesso alle informazioni di rete
- Come possiamo montare un disco (CD, disco esterno) se non abbiamo un desktop environment?
- Visualizzare informazioni di interfaccia (ip, route, altri concetti)
- Network File System (NFS) come disco di rete
- User and Group Management
- Logging and Auditing, etc...

# Outline

- Intro
- Processi Demone
- Il Boot del sistema e Systemd
- Login e Bash Shell
- Gestione del file system
- User and Group Management
- Installazione di pacchetti
- Network File System (NFS)

# Unix System Administration

- Quanto sappiamo utilizzare il terminale? Vi piace o fa troppo old school?
- Il terminale, ossia la shell, è uno strumento fondamentale nel mondo UNIX
- Come avete visto nei corsi di Sistemi Operativi e Architettura di Reti esistono diversi tipi di shell (sh, bash, zsh, csh, ...)
- Ogni tipo ha le proprie peculiarità e funzionalità
- La shell di Bourne (sh) e bash saranno il nostro punto di riferimento durante il corso
- Per rinfrescare la memoria, una guida molto interessante al mondo bash:  
<https://tldp.org/LDP/Bash-Beginners-Guide/html/>

# Linux Distro

**Table 1.1 Most popular general-purpose Linux distributions**

Distribution	Web site	Comments
Arch	archlinux.org	For those who fear not the command line
CentOS	centos.org	Free analog of Red Hat Enterprise
CoreOS	coreos.com	Containers, containers everywhere
Debian	debian.org	Free as in freedom, most GNUish distro
Fedora	fedoraproject.org	Test bed for Red Hat Linux
Kali	kali.org	For penetration testers
Linux Mint	linuxmint.com	Ubuntu-based, desktop-friendly
openSUSE	opensuse.org	Free analog of SUSE Linux Enterprise
openWRT	openwrt.org	Linux for routers and embedded devices
Oracle Linux	oracle.com	Oracle-supported version of RHEL
RancherOS	rancher.com	20MiB, everything in containers
Red Hat Enterprise	redhat.com	Reliable, slow-changing, commercial
Slackware	slackware.com	Grizzled, long-surviving distro
SUSE Linux Enterprise	suse.com	Strong in Europe, multilingual
Ubuntu	ubuntu.com	Cleaned-up version of Debian

Courtesy of Evi Nemeth et al. - Unix and Linux System Administration Handbook

# PROCESSI DEMONE

# Processi Demone

Che cos'è un processo demone? Ne avete mai sentito parlare? Ci sono diverse teorie a riguardo.

“Daemon”, “Daemon Process”, “Background Process” sono termini della lingua inglese a cui noi associamo la traduzione demone, ma in realtà:

Differences between **daemons** and **demons**

Daemon refers to benevolent and noble spirits in Greek mythology, while, demon refers to an evil creature.

# Processi Demone

Un processo demone è un processo che esegue in background e fornisce un servizio:

- a intervalli di tempo prefissati
- in risposta a un evento

I processi demone *non sono associati a un terminale di controllo* e hanno in generale un tempo di vita molto lungo, spesso coincidente col tempo di funzionamento del sistema

Esempio: demone di stampa, web server, ssh, sendmail, cron...



# Processi Demone

Possiamo vedere la lista dei processi in esecuzione sulla nostra macchina utilizzando il comando **ps**, che vedremo più in dettaglio in seguito

**\$ ps axj # BSD syntax**

<b>USER</b>	<b>PID</b>	<b>PPID</b>	<b>PGID</b>	<b>SESS</b>	<b>JOBC</b>	<b>STAT</b>	<b>TT</b>	<b>TIME</b>	<b>COMMAND</b>
<b>root</b>	<b>1</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0 Ss</b>	<b>??</b>	<b>0:15.93</b>		<b>/sbin/launchd</b>
<b>root</b>	<b>67</b>	<b>1</b>	<b>67</b>	<b>0</b>	<b>0 Ss</b>	<b>??</b>	<b>0:00.59</b>		<b>/usr/sbin/syslogd</b>

# Processi Demone

Un processo demone può essere messo in esecuzione:

- al boot del sistema (eseguito da processo `init` o `systemd`)
- dal demone **cron**, che consulta lo script

`/usr/lib/crontab` di sistema

`crontab` di utente (solo System V)

- dal comando **at** (man **at**)
- dal terminale utente

# Processi Demone

Il progetto di un demone deve portare a un funzionamento corretto qualunque sia il modo scelto per la messa in esecuzione del demone, per cui bisogna considerare l'interazione del processo demone con:

- i segnali
- il terminale
- il gruppo di processi

# Sessioni e Process Group

Unix raggruppa i processi in sessioni e process group.

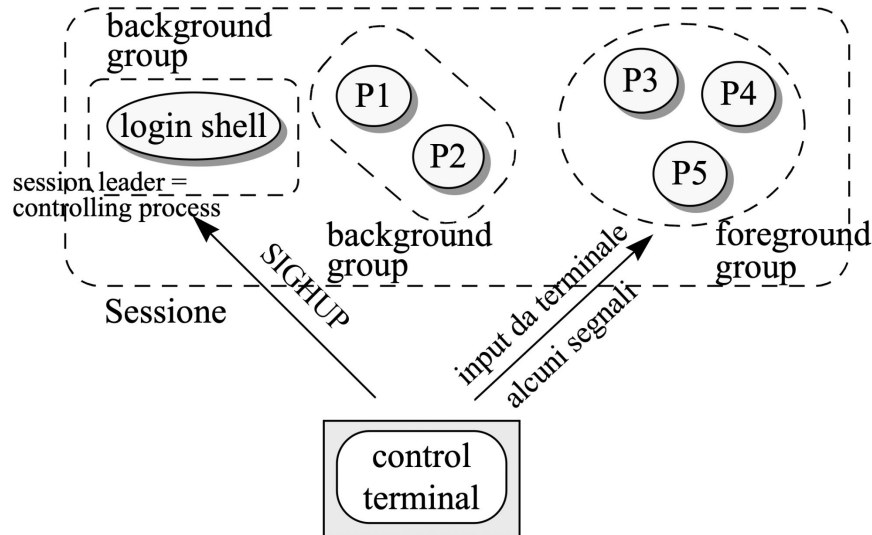
Un process group è un insieme di processi, di solito associati allo stesso job, che possono ricevere segnali dallo stesso terminale. Ogni process group ha un processo “process group leader”.

Una sessione è un insieme di uno o più process group. Ogni sessione ha un processo “session leader” a cui è associato un solo terminale di controllo (terminal device)

# Sessioni e Process Group

esempio:

```
filippo linuxdid ~/ 10> P1 | P2 &  
filippo linuxdid ~/ 11> P3 | P4 | P5
```



# Demoni e Sessioni

Per sganciarsi dal terminale di controllo e diventare effettivamente un demone, un processo deve diventare il *session leader di una nuova sessione che non è associata ad alcun terminale di controllo*.

Per far questo, è necessario che il processo:

- 1) esegua in background
- 2) si sganci dalla sessione corrente

Inoltre, si deve far attenzione a:

- 1) chiusura di tutti i file descriptor eventualmente aperti
- 2) gestione dei segnali
- 3) gestione dell'output, che non è più inviato a un terminale (uso di syslog)

# Processi e terminale di controllo

Per eseguire in background, si deve effettuare una fork e terminare il processo padre:

```
...
pid=fork();
if(pid>0) { /*padre*/
    ...
    exit(0);
} else /* figlio */
```

processi in foreground

ps xj quando il padre e il figlio sono vivi (caso BSD)

PPID	PID	PGID	SID	TPGID	UID	COMMAND
9371	9372	9372	9372	9539	230	-tcsh (tcsh)
9372	<b>9539</b>	<b>9539</b>	9372	<b>9539</b>	230	provaback
9539	9540	9539	9372	<b>9539</b>	230	provaback

processi in background

ps xj dopo il termine del padre

PPID	PID	PGID	SID	TPGID	UID	COMMAND
9371	9372	<b>9372</b>	9372	<b>9372</b>	230	-tcsh (tcsh)
1	9540	9539	9372	<b>9372</b>	230	provaback

# Processi e terminale di controllo

```
...
pid=fork();
if(pid>0) { /*padre*/
    ...
    exit(0);
} else /* figlio */
```

Si noti che, al termine di questa operazione, il processo provaback è l'unico elemento nel gruppo di processi 9539, ma non è il process group leader.

Perdere lo stato di process leader è un passo necessario per potersi sganciare dalla sessione.

**ps xj quando il padre e il figlio sono vivi (caso BSD)**

PPID	PID	PGID	SID	TPGID	UID	COMMAND
9371	9372	9372	9372	9539	230	-tcsh (tcsh)
9372	<b>9539</b>	<b>9539</b>	9372	<b>9539</b>	230	provaback
9539	9540	9539	9372	<b>9539</b>	230	provaback

processi in foreground

processi in background

**ps xj dopo il termine del padre**

PPID	PID	PGID	SID	TPGID	UID	COMMAND
9371	9372	<b>9372</b>	9372	<b>9372</b>	230	-tcsh (tcsh)
1	9540	9539	9372	<b>9372</b>	230	provaback



# Progetto di un demone

- esecuzione di una **fork** con terminazione del padre, per:
  - mettere in background il figlio restituendo il controllo alla shell (considerare il caso in cui il demone venga lanciato da uno script /etc/rc)
  - rendere il figlio non leader di un gruppo (prerequisito per setsid())
- esecuzione di **setsid()** per **sganciare il processo dal terminale di controllo e staccarsi dal gruppo del processo padre** (il processo diventa leader di una nuova sessione non collegata a nessun terminale)
- **chdir("/")** per cambiare la directory ed andare in /, per non bloccare un eventuale filesystem montato, oppure **chdir("/var/spool")** nel caso di demoni che lavorino in specifiche directory del filesystem

# Progetto di un demone

- file mode creation mask a 0 ( **umask(0)** )
- chiusura di tutti i file descriptor
- Installare un **gestore** del SIGTERM per una terminazione graduale e controllata delle sue operazioni (termine processi: SIGTERM e dopo alcuni secondi SIGKILL)
- non lasciare figli zombie:
  - raccogliere lo stato di terminazione oppure
  - generare figli segnalando al kernel di non essere interessati allo stato di ritorno dei figli stessi

# Progetto di un demone

```
#include <stdio.h>
#include <sys/types.h>

int daemon_init(void)
{
    pid_t pid;
    int i;

    if ( (pid=fork()) <0) return (-1);

    if(pid>0)
        exit(0);
    else {
        /* codice eseguito dal figlio */

        setsid();

        chdir("/");
```

# Progetto di un demone

```
    umask(0);

    for(i=0;i<_NFILE;i++) close(i);
    /* _NFILE è una costante che indica
       il max num di fd per processo */
    // non fa niente
    return(0);
}
}
```

# Progetto di un demone

Nel progetto di un demone bisogna considerare le differenze dovute alle diverse versioni Unix.

Per esempio, in **BSD** bisognerebbe eseguire

```
#ifdef SIGTTOU
    signal(SIGTTOU, SIG_IGN);
#endif
```

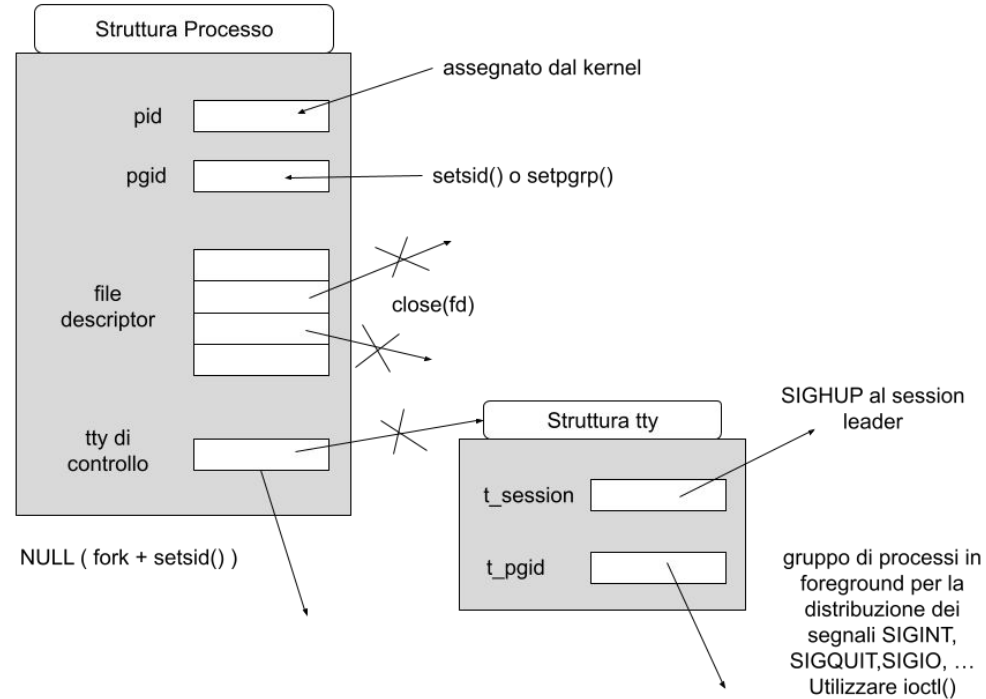
```
#ifdef SIGTTIN
    signal(SIGTTIN, SIG_IGN);
#endif
```

al posto della `setsid()`:

`setpgrp(0, getpid())` per creare un nuovo gruppo di cui il demone è leader aprire il file `/dev/tty` e inviargli un `TIOCNTTY` per scollegarsi dal terminale

```
ioctl(fd, TIOCNTTY, (char *) 0);
```

# Struttura dati del Kernel



# **IL BOOT DEL SISTEMA E LA GESTIONE DEI SERVIZI**

# Il processo di boot - Sys V (caso storico)

Vediamo ora come vengono mandati in esecuzione al momento del boot i demoni di sistema. Abbiamo che al momento del boot vengono eseguiti i seguenti passi:

- 1) Il boot loader legge l'immagine del kernel dal disco fisso (in una posizione pre-configurata) e la carica in memoria
- 2) Il kernel lancia il processo init
- 3) Init inizializza la parte userspace del sistema secondo le modalità definite nel file di configurazione `/etc/inittab`

In particolare, `/etc/inittab` definisce diverse modalità di funzionamento del sistema, che prendono il nome di run level.



# Il processo di boot - Sys V (caso storico)

La configurazione standard di init prevede i seguenti run level:

0	System halt
1	Single user mode
2	Single user mode con accesso alla rete
3	Modalità multiutente con login testuale
4	Non definita
5	Modalità multiutente con login grafico
6	Shutdown e reboot

A seconda del run level scelto dall'utente (è possibile specificare un run level diverso da quello predefinito al momento del boot passando un apposito parametro al kernel), init lancia i servizi di sistema configurati nella directory `/etc/rc.d/rc{RUNLEVEL}.d`.

I servizi di sistema includono il system log daemon, il power management daemon, e altri demoni che gestiscono servizi di rete come HTTP, E-mail e DNS.

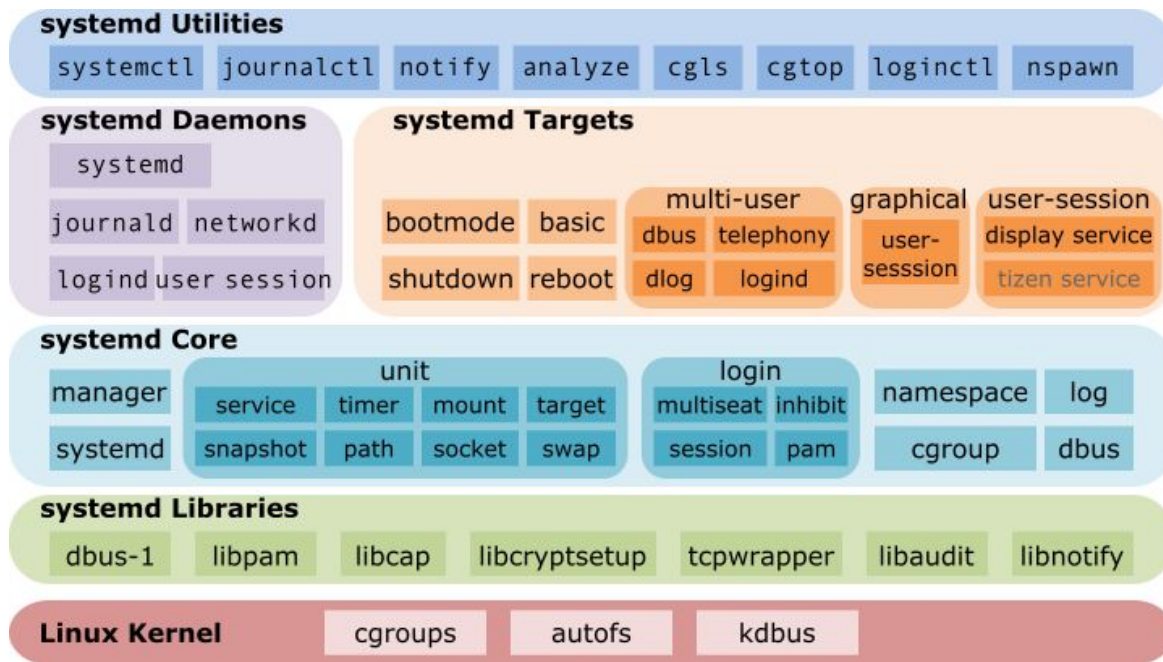
# Il processo di boot - systemd

La quasi totalità delle distribuzioni di Linux utilizza **systemd** per le operazioni connesse al boot. **systemd** è utile per svolgere diverse operazioni:

- offre un supporto molto migliore alla gestione delle dipendenze tra i servizi;
- permette la messa in esecuzione parallela di diversi servizi, anche quelli interdipendenti, portando a una notevole riduzione dei tempi di avvio del sistema;
- offre una migliore integrazione con udev, il sistema di gestione dei dispositivi hardware di Linux;
- fornisce un sottosistema “leggero” di gestione dei log pensato per sostituire syslog, il tradizionale demone di gestione dei log di sistema pensato per ambienti server, in macchine desktop/laptop;
- supera la soluzione runlevel di Sys V.

Per ulteriori informazioni, si consulti il sito Web: <http://freedesktop.org/wiki/Software/systemd> o la pagina di wikipedia: <https://en.wikipedia.org/wiki/Systemd>

# Systemd - Architettura



Courtesy of Shmuel Csaba Otto Traian, CC BY-SA 3.0,  
<https://commons.wikimedia.org/w/index.php?curid=28698339>

# Systemd per la gestione di servizi

Una delle funzionalità più importanti di systemd è la gestione dei servizi. Con systemd viene superato il vecchio sistema rc, rendendo disponibile un'interfaccia più gradevole all'utente.

La gestione di un servizio riguarda la gestione del ciclo di vita di un servizio, gestito tramite il comando **systemctl**.

**Systemctl** prende come primo argomento un sotto-comando, dopodiché i parametri successivi sono relativi al sotto-comando.

Visualizzare i servizi  
abilitati e attivi

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

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION	
accounts-daemon.service	loaded	active	running	Accounts Service	>
apparmor.service	loaded	active	exited	Load AppArmor profiles	>
apport.service	loaded	active	exited	LSB: automatic crash report generati	>
atd.service	loaded	active	running	Deferred execution scheduler	>
blk-availability.service	loaded	active	exited	Availability of block devices	

# Systemd per la gestione di servizi

Un altro sotto-comando utilizzato è quello per controllare lo stato di un servizio, ad esempio **cups...**

```
$ sudo systemctl status -l cups # [cups.service]
```

```
# abilitare lo startup un servizio al boot startup
```

```
$ sudo systemctl enable cups.service
```

```
# disabilitare lo startup di un servizio al boot
```

```
$ sudo systemctl disable cups.service
```

```
# attivare un servizio
```

```
$ sudo systemctl start cups.service # analogamente i comandi di stop/restart
```

# Systemd per la gestione di servizi

## Commonly used systemctl subcommands

Subcommand	Function
<b>list-unit-files</b> [ <i>pattern</i> ]	Shows installed units; optionally matching <i>pattern</i>
<b>enable</b> <i>unit</i>	Enables <i>unit</i> to activate at boot
<b>disable</b> <i>unit</i>	Prevents <i>unit</i> from activating at boot
<b>isolate</b> <i>target</i>	Changes operating mode to <i>target</i>
<b>start</b> <i>unit</i>	Activates <i>unit</i> immediately
<b>stop</b> <i>unit</i>	Deactivates <i>unit</i> immediately
<b>restart</b> <i>unit</i>	Restarts (or starts, if not running) <i>unit</i> immediately
<b>status</b> <i>unit</i>	Shows <i>unit</i> 's status and recent log entries
<b>kill</b> <i>pattern</i>	Sends a signal to units matching <i>pattern</i>
<b>reboot</b>	Reboots the computer
<b>daemon-reload</b>	Reloads unit files and <b>systemd</b> configuration

Courtesy of Evi Nemeth et al. Commonly used systemctl subcommands

# Systemd per la gestione di servizi

Esempio, voglio controllare lo stato del web server nginx in esecuzione sulla mia macchina:

```
$ sudo systemctl status nginx.service
```

```
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2021-12-15 14:59:28 CET; 1 day 1h ago
     Docs: man:nginx(8)
  Main PID: 1991 (nginx)
    Tasks: 2 (limit: 1136)
   Memory: 1.8M
    CGroup: /system.slice/nginx.service
            └─1991 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
               └─1992 nginx: worker process
```

```
Dec 15 14:59:27 primary systemd[1]: Starting A high performance web server and a reverse proxy server...
```

```
Dec 15 14:59:28 primary systemd[1]: Started A high performance web server and a reverse proxy server.
```

# **LOGIN E SHELL**



# Login

Una volta effettuato il boot, l'utente può effettuare il login, tipicamente in modalità testuale per installazioni di tipo server e in modalità grafica per installazioni di tipo desktop.

In modalità testuale, all'utente viene immediatamente resa disponibile una shell di login.

In modalità grafica, viene lanciato l'ambiente grafico (X-Windows/Wayland) con le proprie configurazioni (desktop environment, ecc...). In questo caso, per poter accedere a una shell, l'utente deve lanciare un emulatore di terminale (ad esempio konsole, gnome-terminal, xterm, rxvt, ecc...).

Una volta aperta una shell, Linux mette a disposizione molti comandi per l'amministrazione di sistema. Noi vedremo alcuni dei comandi principali.

# BASH: File di configurazione

Bash implementa una gestione più flessibile dei file di configurazione rispetto alla tradizionale shell di Bourne.

Un'ottima guida alle funzionalità di shell è disponibile gratuitamente all'indirizzo:  
<https://tldp.org/LDP/Bash-Beginners-Guide/html/>

Allo startup Bash legge una serie di file, in cui vengono tenute le configurazioni a livello di sistema e/o di singolo utente.

C'è una differenza tra shell di login (che hanno il compito di caricare le variabili di ambiente) e shell lanciate successivamente (che ricevono le variabili di ambiente dalla shell di login).

# BASH: File di configurazione

Per le shell di login vengono letti, in ordine, i seguenti file:

<b>/etc/profile</b>	Configurazione a livello di sistema. Alcune distribuzioni di Linux considerano anche i file <code>/etc/profile.d/*</code>
<b>~/.bash_profile, ~/.bash_login o ~/.profile</b>	Configurazione a livello di singolo utente. Viene considerato solo il primo file esistente e leggibile tra questi tre.
<b>~/.bash_logout</b>	Configurazione a livello di singolo utente. Viene caricato al momento del logout.

# BASH: File di configurazione

Per le shell *non di login*, ossia tutte le shell lanciate da una shell di login, viene considerato esclusivamente il file di configurazione:

<code>~/.bashrc</code>	Configurazione a livello di singolo utente. Molto spesso questo file viene configurato per includere le opzioni specificate a livello di sistema in <code>/etc/bashrc</code> .
------------------------	---

Si noti che nella configurazione di default di molte distribuzioni `~/.bashrc` viene spesso incluso da `~/.profile` o `~/.bash_profile` tramite un comando simile a:

```
if [ -f ~/.bashrc ]; then
    . ~/.bashrc;
fi
```

`[ ]` sono l'equivalente di test

# Aliases

Gli alias permettono la sostituzione di una stringa al posto della prima parola di un semplice comando.

Bash mantiene una lista di alias che può essere visualizzata e modificata dinamicamente con il comando `alias`.

```
$ alias
```

```
alias ll='ls -l'  
alias ls='exa'
```

Per creare un alias:

```
$ alias rf="rm -rf"
```

# Debugging

Le opzioni `-x` e `-v` semplificano il debugging di uno script. È importante acquisire una buona manualità con l'utilizzo di questi strumenti, che permettono di facilitare moltissimo l'individuazione degli errori più comuni.

**set -v**     Stampa gli input che vengono forniti alla shell

**set -x**     Stampa i comandi che vengono eseguiti

Questi comandi si possono applicare all'esecuzione di un intero script, lanciando la shell con l'opzione `-x`:

```
$ bash -x script.sh
```

# Debugging

Alternativamente, è possibile abilitare e disabilitare i messaggi di debug solo per una parte limitata dello script:

```
#!/bin/bash
...
set -x # attiva il debugging
find . -iname "*pippo*" -exec rm -f {} \;
set +x # ferma il debugging
...
```

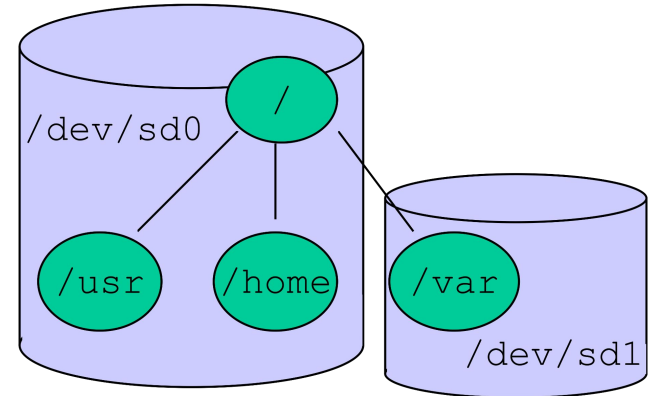
# **FILE SYSTEM**



# Gestion del File System

- L'insieme delle risorse gestite da un sistema Unix è rappresentato da un insieme di file system, organizzati all'interno della stessa struttura logica, il FILESTORE
- Un file system corrisponde a un dispositivo fisico (es. un disco oppure una partizione di disco, /dev/sda0...)
- Più dispositivi fisici possono essere aggregati in modo omogeneo all'interno della stessa struttura logica (il Filestore)

Filesystem	mounted on
/dev/sd0a	/
/dev/sd0h	/usr
/dev/sd0g	/home
/dev/sd1c	/var



# Il File System di Linux

/bin	Eseguibili di base
/boot	File per il boot del sistema (kernel e bootloader)
/dev	File che rappresentano le periferiche del sistema
/etc	File di configurazione a livello di sistema
/home	Cartelle personali degli utenti
/media e/o /mnt	Per montare dischi e/o chiavette USB
/opt	Per l'installazione di programmi opzionali o secondari
/proc e /sys	File speciali per il monitoraggio e/o la riconfigurazione on-the-fly del sistema

# Il File System di Linux

/root	Cartella dell'utente root
/sbin	Eseguibili di base riservati all'amministratore
/tmp	File temporanei
/usr	Contiene la maggior parte dei programmi installati sul sistema
/var	Dati di dimensione variabile ad uso di vari servizi (es. server Web, DNS, Email)

# Comando du

Il comando **du** permette di visualizzare la quantità usata di spazio su disco da una determinata directory e dalle relative subdirectory.

Se non viene specificata una directory in particolare, du considera la directory corrente.

```
$ du -k --max-depth=1 .  
 35796    ./git  
   12     ./vscode  
   12     ./bin  
112196    ./examples  
   152    ./lib  
   52     ./spec  
14744     ./vendor  
163020    .
```

# Comando du

Alcune opzioni importanti:

- k** stampa la dimensione dei dati in KB
- m** stampa la dimensione dei dati in MB
- h** stampa la dimensione in formato più leggibile, usando l'opportuna unità di misura (KB, MB, GB) a seconda delle dimensioni del file
- max-depth=N** limita la stampa a video dei risultati ai primi N sottolivelli della directory (tutti i sottolivelli vengono tenuti in considerazione per il calcolo della dimensione)

# Comando df

Il comando **df** permette di visualizzare, partizione per partizione, l'ammontare di spazio libero su disco. In particolare, **df** fornisce informazioni sulla dimensione, spazio libero, spazio allocato e punto di mount di ciascuna partizione in uso.

```
$ df
Filesystem      1K-blocks      Used   Available Use% Mounted on
udev            485108          0     485108    0% /dev
tmpfs           100488         876       99612    1% /run
/dev/vda1       4901996 1636592    3249020   34% /
tmpfs           502432          0     502432    0% /dev/shm
...
/dev/vda15      106858         5321     101537    5% /boot/efi
tmpfs           100484          0     100484    0% /run/user/1000
:/Users/filippo 1048576000      0 1048576000    0% /home/ubuntu/Home
```

# mount

Il comando **mount** permette di inserire (o, in gergo tecnico, “montare”) i file contenuti su un determinato dispositivo (partizione, CD/DVD, USB stick, ecc.) all’interno del file system. **Questo comando è di fondamentale importanza!**

Si ricordi che Unix fornisce un file system unico e gerarchicamente organizzato a partire dalla root directory (ovverosia la directory */*). *In questo contesto, è possibile che alcune parti del file system risiedano su diversi dispositivi e/o vengano montate e smontate all’occorrenza.*

Ad esempio, per montare il contenuto della partizione */dev/sda5* all’interno della directory */mnt/pippo* si usa il comando:

```
$ mount /dev/sda5 /mnt/pippo
```

# mount

Quando invocato senza opzioni e/o parametri, il comando **mount** si limita a visualizzare la lista dei dispositivi attualmente “montati” nel file system.

```
$ mount
/dev/disk1s5s1 on / (apfs, sealed, local, read-only, journaled)
devfs on /dev (devfs, local, nobrowse)
/dev/disk1s4 on /System/Volumes/VM (apfs, local, noexec, journaled, noatime,
nobrowse)
/dev/disk1s2 on /System/Volumes/Preboot (apfs, local, journaled, nobrowse)
/dev/disk1s6 on /System/Volumes/Update (apfs, local, journaled, nobrowse)
/dev/disk1s1 on /System/Volumes/Data (apfs, local, journaled, nobrowse)
map auto_home on /System/Volumes/Data/home (autofs, automounted, nobrowse)
/dev/disk1s5 on /System/Volumes/Update/mnt1 (apfs, sealed, local, journaled,
nobrowse)
```



# Mount di un CD/DVD

Ad esempio, vogliamo fare il mount di un CD o di un DVD in Linux. Il primo passo è identificare il file device (/dev/sd...) utilizzato dal CD. Possiamo utilizzare il comando blkid.

```
$ blkid  
/dev/sr0: UUID:"....." TYPE="ISO9660"
```

**ISO9660** è il formato utilizzato da un CD/DVD e **/dev/sr0** il file device che dobbiamo montare.

# Mount di un CD/DVD

Prima di lanciare il comando mount dobbiamo però creare una directory di appoggio

```
$ sudo mkdir /mnt/cdrom
```

e infine montare il device

```
$ sudo mount -t iso9660 /dev/sr0 /mnt/cdrom
```

Si può inoltre visualizzare lo stato dell'operazione utilizzando il comando mount senza parametri.

**Perchè sudo?**

# sudo

L'utility **sudo** permette ad un utente normale di lanciare un singolo specifico comando come se fosse un amministratore di sistema. **sudo** è un comando di particolare importanza nel mondo UNIX.

Cosa fa il comando wc?

```
$ sudo wc /etc/sudoers
```

```
Password:
```

```
60      228      1563 /etc/sudoers
```

```
$ sudo whoami # eseguendo il comando whoami mi viene ritornato sudo
root
```

```
$ man whoami
```

```
DESCRIPTION
```

```
Print the user name associated with the current effective user ID.
```

```
Same as id -un.
```

# sudo

Per poter permettere agli utenti di acquisire temporaneamente i privilegi di amministratore di sistema, sudo deve girare con i privilegi di root (owner root + suid), per cui è indispensabile configurarlo correttamente.

A questo proposito, le distribuzioni offrono una configurazione di default piuttosto sicura. Solo gli utenti del gruppo wheel possono eseguire comandi tramite sudo. Si può inoltre restringere il set di comandi che è possibile eseguire tramite sudo modificando appositamente il file di configurazione **/etc/sudoers**.

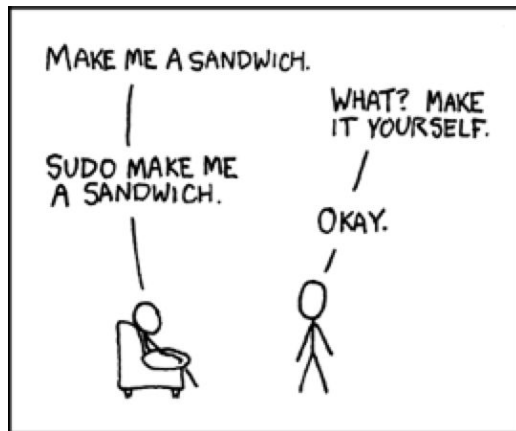
La modalità di amministrazione tramite login come utente non privilegiato e l'uso di sudo per compiti specifici è attualmente considerata la più sicura in ambiente Unix/Linux.

Alcune distribuzioni di Linux, ad esempio Ubuntu, non permettono nemmeno all'utente di effettuare il login come amministratore di sistema, e propongono invece sudo come l'unico meccanismo disponibile per l'amministrazione di sistema.

# sudo

Si consiglia caldamente di familiarizzare con sudo e di usarlo quotidianamente per l'amministrazione del proprio sistema.

L'uso di sudo per l'amministrazione di macchine Unix è diventato talmente diffuso da far entrare prepotentemente questa utility nella computer geek culture:



Vignetta presa dal sito Web xkcd: <http://xkcd.com/149/>

# sudo

La configurazione del file **/etc/sudoers** è un'operazione complessa, specialmente se si vogliono definire delle policy speciali.

L'operazione più semplice è l'assegnazione dei permessi di amministratore a un utente. A tal fine si deve modificare il file **/etc/sudoers** con un editor apposito, ad esempio **visudo**.

```
$ sudo visudo /etc/sudoers # apro il file
```

```
# aggiungo i privilegi di amministratore all'utente filippo
```

```
filippo ALL=(ALL) ALL
```

# sudo

Altri esempi di configurazione di default in /etc/sudoers

```
# Host alias specification
```

```
# User alias specification
```

```
# User privilege specification
```

```
root    ALL=(ALL:ALL) ALL
```

```
# Members of the admin group may gain root privileges
```

```
%admin  ALL=(ALL) ALL
```

Esempio interessante



```
# Allow members of group sudo to execute any command
```

```
%sudo   ALL=(ALL:ALL) ALL
```

# Sudo - Logs

Sudo tiene traccia delle operazioni svolte attraverso un log, in cui vengono registrati i comandi eseguiti insieme ad altre informazioni utili.

Questo log può essere memorizzato in un file (**/var/log/auth.log**) o gestito tramite syslog, per registrare le operazioni in un server sicuro.

Esempio:

```
sudo tail /var/log/auth.log  
Dec 16 17:46:34 primary sudo:    ubuntu : TTY=pts/0 ; PWD=/home/ubuntu  
; USER=root ; COMMAND=/usr/bin/cat /etc/sudoers
```



# Oltre l'utilizzo di sudo

Il problema della sicurezza dei sistemi è *sempre più importante e fondamentale*.

Negli ultimi anni (decenni), sono apparse soluzioni con un grado di complessità più elevato rispetto al comando sudo, che superano lo Standard Access Control Model di UNIX / Linux.

Esempi di queste soluzioni sono **Mandatory Access Control (MAC)** e **Role-Based access control (RBAC)**, che non vedremo durante questo corso.

Si cita però l'implementazione di Linux MAC **SeLinux**, una delle più vecchie implementazioni prodotte dalla U.S. National Security Agency (NSA). SeLinux è supportato da diverse distribuzioni Linux come RedHat, Centos e Fedora.

Per approfondire queste tematiche si consulti l'eventuale materiale aggiuntivo.

# Gestione degli utenti

Linux, come la maggior parte dei sistemi moderni, è un sistema multiutente. Le informazioni associate agli utenti sono registrate nel file **/etc/passwd**

```
pltfpp@linuxdid:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

**/etc/group** invece mantiene le informazioni associate a ogni gruppo definito nel sistema

```
pltfpp@linuxdid:~$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
```

# Gestione degli utenti

Per aggiornare la password dell'utente si utilizza il comando **passwd**

L'utente root può anche aggiornare la password degli altri utenti, specificando come parametro al comando `passwd` lo username per l'utente di cui si vuole aggiornare la password

```
$ passwd <nome_utente>
```

# Aggiunta di un nuovo utente

Per aggiungere un nuovo utente va utilizzato il comando **useradd**, supportato da molte distribuzioni Linux. Alcune distribuzioni come Debian mettono a disposizione comandi di più alto livello che wrappano **useradd**.

**useradd** è configurabile con diverse opzioni, quelle di default sono visibili in */etc/default/useradd*.

La sintassi del comando è la seguente

```
$ useradd <nome_utente>
```

# Aggiunta di un nuovo utente

Ad esempio:

```
$ sudo useradd -c "Pippos" -d /home/pippo -m -s  
/bin/bash pippo
```

Crea un nuovo **utente** pippo, aggiungendo una entry in */etc/passwd* e nel file shadow corrispondente. **-m** crea una home directory se non già presente e viene assegnata */bin/bash* come shell di default.

Va poi assegnata una password all'account appena creato:

```
$ sudo passwd pippo
```

# Eliminazione di un utente

Per rimuovere l'utente appena creato si può utilizzare il comando **userdel** con i privilegi di amministratore:

```
$ sudo userdel -r pippo
```

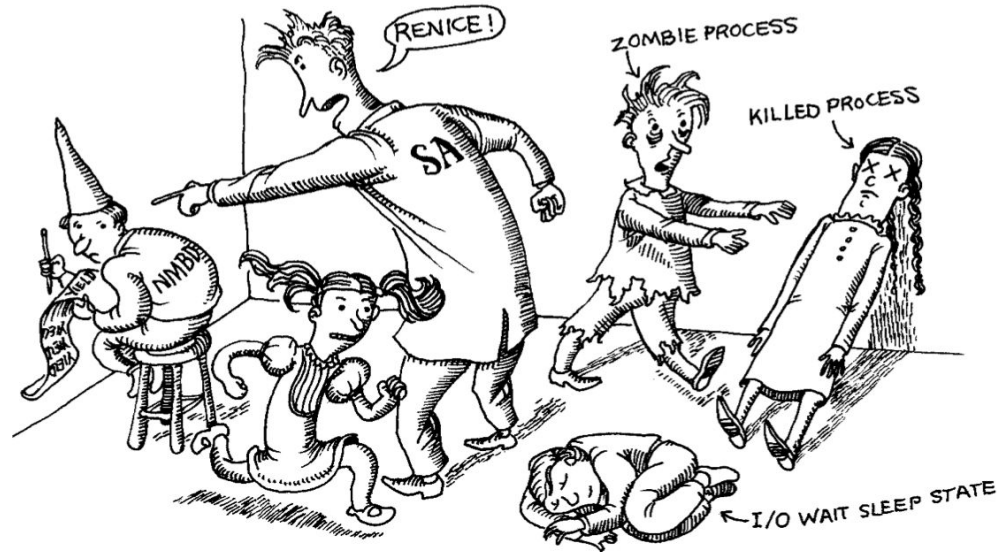
L'opzione -r permette di rimuovere la home directory associata all'utente. Inoltre bisognerebbe verificare la presenza di file associati all'utente all'interno del filesystem:

```
$ sudo find / -xdev -nouser
```

Trova una lista dei file associati all'utente rimosso.

# I processi

## *Process Control*



Courtesy of Evi Nemeth et al.

# top

Il comando top fornisce una visione dinamica in real-time del sistema corrente. Esso visualizza continuamente informazioni sull'utilizzo del sistema (memoria fisica e virtuale, CPU, ecc.) e sui processi che usano la maggiore share di CPU.

```
$ top
```

```
top - 17:57:02 up 14:49, 2 users, load average: 0.00, 0.00, 0.00
```

```
Tasks: 106 total, 1 running, 105 sleeping, 0 stopped, 0 zombie
```

```
%Cpu(s): 0.0 us, 0.3 sy, 0.0 ni, 99.3 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st
```

```
MiB Mem : 981.3 total, 228.6 free, 144.3 used, 608.4 buff/cache
```

```
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 683.3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	172096	14060	8220	S	0.0	1.4	0:09.50	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.05	kthreadd

```
----
```



# ps

Il comando ps mostra la lista dei processi attualmente in esecuzione. Se si vuole un aggiornamento continuo e periodico dello stato dei processi correnti, si usi il comando top.

Ecco l'output del comando:

```
$ ps aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	1.3	172096	14060	?	Ss	03:07	0:09	/sbin/init
root	2	0.0	0.0	0	0	?	S	03:07	0:00	[kthreadd]
root	3	0.0	0.0	0	0	?	I<	03:07	0:00	[rcu_gp]
root	4	0.0	0.0	0	0	?	I<	03:07	0:00	[rcu_par_gp]
root	6	0.0	0.0	0	0	?	I<	03:07	0:00	[kworker/0:0H-kblockd]

# ps

Alcune opzioni importanti:

- a** Mostra anche i processi degli altri utenti
- u** Fornisci nome dell'utente che ha lanciato il processo e l'ora in cui il processo è stato lanciato
- x** Mostra anche i processi senza terminale di controllo (processi demoni)

Attenzione: si ricordi di non usare il trattino (-) per specificare le opzioni del comando ps, in quanto ps adotta la sintassi di tipo “extended BSD” che non prevede l'uso di trattini

# /PROC Filesystem

I comandi `ps` e `top` leggono le informazioni sullo stato dei processi dalla directory `/proc`.

`/proc` è un pseudo-filesystem in cui il kernel espone diverse informazioni sullo stato del sistema.

**Table 4.4 Process information files in Linux `/proc` (numbered subdirectories)**

File	Contents
<code>cgroup</code>	The control groups to which the process belongs
<code>cmd</code>	Command or program the process is executing
<code>cmdline</code> <sup>a</sup>	Complete command line of the process (null-separated)
<code>cwd</code>	Symbolic link to the process's current directory
<code>environ</code>	The process's environment variables (null-separated)
<code>exe</code>	Symbolic link to the file being executed
<code>fd</code>	Subdirectory containing links for each open file descriptor
<code>fdinfo</code>	Subdirectory containing further info for each open file descriptor
<code>maps</code>	Memory mapping information (shared segments, libraries, etc )
<code>ns</code>	Subdirectory with links to each namespace used by the process.
<code>root</code>	Symbolic link to the process's root directory (set with <code>chroot</code> )
<code>stat</code>	General process status information (best decoded with <code>ps</code> )
<code>statm</code>	Memory usage information

a. Might be unavailable if the process is swapped out of memory

# Comandi nice e renice

Con i comandi **nice** e **renice** è possibile modificare la priorità assegnata ai processi. La ***nice***ness di un process indica quanto questo è “cortese” nei confronti degli altri processi. *Una nice elevata quindi equivale a un basso livello di priorità.*

La scala va da -19 (-20 per Linux) a 20 ed è consultabile utilizzando le utility ps e top. La nice può essere specificata al momento del lancio (utilizzando il comando **nice**) o modificata in seguito (con **renice**) utilizzando i permessi di amministratore.

La pratica di modificare la niceness dei processi è caduta in disuso. Nei moderni sistemi operativi lo scheduler è abbastanza *intelligente* per fare bene il suo lavoro.

# Comandi nice e renice

Per esempio, in MacOS la niceness di tutti i processi è settata a 0...

Vediamo:

```
# specifico di visualizzare la niceness (ni) in output  
$ ps aux -o pid,ni
```

In linux invece viene specificata ancora la niceness di alcuni processi

```
# specifico di visualizzare la niceness (ni)  
# non-BSD syntax -e equivalente a aux  
$ ps -eo pid,ni,cmd
```

# pgrep

pgrep restituisce i pid (process id) dei processi che corrispondono alle caratteristiche (nome processo, utente, ecc.) richieste.

Ad esempio, per ottenere il pid del processo chrome:

```
$ pgrep chrome
```

```
7456
```

```
7548
```

pgrep come grep è case sensitive... Come posso fare per lanciare una ricerca ignorando il case dei caratteri?

Per ottenere invece il pid del processo demone sshd appartenente all'utente root:

```
$ pgrep -u root sshd
```

```
3488
```

# pgrep

**pgrep** presenta la stessa interfaccia domandi di **pgrep**, ma anziché stampare a video il pid dei processi, invia loro un segnale di terminazione (di default SIGTERM).

\$ **pgrep chrome**

# wget

**GNU wget** è una utility per il download non-interattivo di file dal Web. wget supporta i protocolli HTTP, HTTPS e FTP. **wget** è utilissimo per velocizzare e/o automatizzare molti compiti amministrativi che richiedono l'uso della rete.

A terminal window titled 'filippo — filippo@MacBook-Pro-di-Filippo — ~ — zsh — 88x26'. The terminal shows the execution of the command 'wget https://cache.ruby-lang.org/pub/ruby/3.0/ruby-3.0.3.tar.gz'. The output displays the URL, IP addresses (151.101.1.178, 151.101.65.178, 151.101.129.178), connection status, HTTP response (200 OK), file length (20242729 bytes), and the file being saved ('ruby-3.0.3.tar.gz.1'). A progress bar shows 100% completion. The final line indicates the file was saved at 8.27 MB/s in 2.3 seconds.

```
+ ~ wget https://cache.ruby-lang.org/pub/ruby/3.0/ruby-3.0.3.tar.gz
--2021-12-16 18:11:24-- https://cache.ruby-lang.org/pub/ruby/3.0/ruby-3.0.3.tar.gz
Resolving cache.ruby-lang.org (cache.ruby-lang.org)... 151.101.1.178, 151.101.65.178, 15
1.101.129.178, ...
Connecting to cache.ruby-lang.org (cache.ruby-lang.org)|151.101.1.178|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20242729 (19M) [application/x-tar]
Saving to: 'ruby-3.0.3.tar.gz.1'

ruby-3.0.3.tar.gz.1 100%[=====] 19,30M 8,27MB/s in 2,3s

2021-12-16 18:11:26 (8,27 MB/s) - 'ruby-3.0.3.tar.gz.1' saved [20242729/20242729]

[~]
[~]
[~]
[~]
[~]
[~]
[~]
[~]
[~]
[~]
```



# lsof

**lsof** (sta per list open files) è un comando molto importante anche se poco conosciuto che permette di ottenere informazioni sui file aperti dai processi in esecuzione sul sistema.

Ad esempio, per trovare la lista di processi che stanno usando il file `zuo2018.pdf` basta eseguire:

```
$ lsof zuo2018.pdf
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
Preview	26984	filippo	txt		REG	1,6	842559	15243461

`zuo2018.pdf`

# lsof

**lsof** mette a disposizione diverse opzioni, alcuni esempi sono

lista dei file aperti da un processo

```
$ lsof -p <pid>
```

lista dei file aperti dall'utente

```
$ lsof -u <user>
```

Poiché tutto in Unix è un file, lsof permette anche di ottenere informazioni su pipe, socket, periferiche, ecc.

# Isof

Poiché tutto in Unix è un file, **Isof** permette anche di ottenere informazioni su pipe, socket, periferiche, ecc.

L'opzione -i :N permette di vedere quali processi stanno usando la porta N

```
[→ ~ lsof -i :22
COMMAND      PID      USER    FD  TYPE             DEVICE SIZE/OFF NODE NAME
multipass    27545  filippo  10u  IPv4 0x5a18314fbe53f39b      0t0  TCP 192.168.64.1:59092-
>192.168.64.5:ssh (ESTABLISHED)
→ ~ █
```

Quando invocato senza parametri, Isof mostra la lista di tutti i file aperti da tutti i processi attualmente in esecuzione.... **provare...**

# Variabili d'ambiente

Per conoscere la lista e il valore delle variabili d'ambiente della shell possiamo utilizzare il comando **printenv**.

```
[→ ~ printenv | head -15
__CFBundleIdentifier=com.apple.Terminal
TMPDIR=/var/folders/yb/9lk8l3vx7p9d7wzmx8shrbkw0000gn/T/
XPC_FLAGS=0x0
LaunchInstanceID=3067AD1D-5A36-436F-A913-2DE3992788D4
TERM=xterm-256color
DISPLAY=/private/tmp/com.apple.launchd.F8fNwefbu2/org.macosforge.xquartz:0
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.V3gl0qGgwK/Listeners
SECURITYSESSIONID=186aa
XPC_SERVICE_NAME=0
TERM_PROGRAM=Apple_Terminal
TERM_PROGRAM_VERSION=440
TERM_SESSION_ID=F339CCEE-8572-4655-BE3C-16F3E95388D3
SHELL=/bin/zsh
HOME=/Users/filippo
LOGNAME=filippo
```

```
→ ~ █
```

# gzip

I comandi **gzip** e **gunzip** permettono rispettivamente di comprimere e decomprimere un file usando un algoritmo standard di tipo Lempel-Ziv (LZ77).

Ad esempio, il seguente comando comprime il file `pippo.txt` generando il file compresso `pippo.txt.gz`:

```
$ gzip pippo.txt
```

Se non viene specificato esplicitamente il nome da usare per il file compresso, `gzip` ne genera uno automaticamente, aggiungendo il suffisso `.gz` al nome del file originale.

Attenzione: una volta salvato il file compresso / decompresso, `gzip` e `gunzip` cancellano il file originale passatogli a riga di comando.

Il seguente comando ripristina il file `pippo.txt`, a partire dal file compresso `pippo.txt.gz`:

```
$ gunzip pippo.txt.gz
```

# zip e unzip

Esistono anche altri comandi simili a gzip che permettono di comprimere file o cartelle. Tra questi troviamo sicuramente zip e unzip, disponibili in diverse versioni di Linux e UNIX (MacOS). La sintassi di utilizzo è simile

```
$ zip foo.zip foo
```

Comprime il file foo in un archivio chiamato foo.zip

Se invece vogliamo comprimere una intera cartella, comprese le sue sottocartelle:

```
$ zip -r compressed_folder.zip folder
```

dove -r specifica l'opzione ricorsiva. Che va decompressa con il relativo comando unzip

```
$ unzip compressed_folder.zip
```

# tar

Un altro comando utilissimo nel mondo Linux è **tar**. Il comando tar è un utility per l'archiviazione, che consente di salvare il contenuto di uno o più file (o directory) in un singolo file archivio.

```
$ tar cvf pippus.tar pippo.txt pluto.txt
```

creo un archivio con i file pippo.txt e pluto.txt:

Per creare un archivio che contenga l'intera directory codice (con le rispettive sottodirectory):

```
$ tar cvf pippo.tar codice
```

Per stampare la lista dei file inclusi nell'archivio:

```
$ tar tvf pippo.tar
```

Per estrarre il contenuto dell'archivio:

```
$ tar xvf pippo.tar
```

# tar

Il comando tar permette anche di operare su archivi compressi tramite gzip o bzip2, tramite le opzioni z e j.

Per creare un archivio compresso tramite gzip che contenga l'intera directory codice (con le rispettive sottodirectory):

```
$ tar cvfz pippo.tar.gz codice
```

Per estrarre il contenuto dell'archivio:

```
$ tar xvfz pippo.tar.gz
```

Per utilizzare bzip2 avremmo dovuto specificare l'opzione j al posto di z.



# tar

È convenzione usare i seguenti suffissi (o estensioni, che dir si voglia) per i nomi di file di archivio:

.tar	Archivio tar non compresso
.tar.gz o .tgz	Archivio tar compresso tramite gzip
.tar.bz2 o .tbz	Archivio tar compresso tramite bzip2

# tar

-n	Stampa il nome dell'host
-r	Stampa la release del kernel in uso
-m, -p e -i	Stampano informazioni sulla piattaforma hardware del sistema
-s	Stampa il nome del kernel (comportamento di default in caso nessun opzione sia stata specificata)
-o	Stampa il nome del sistema operativo
-a	Stampa tutte le informazioni elencate sopra

# id

Il comando id stampa informazioni sull'utente. Alcune opzioni:

<b>-u</b>	Stampa lo user id
<b>-g</b>	Stampa l'id del gruppo principale di cui fa parte l'utente
<b>-G</b>	Stampa gli id di tutti i gruppi di cui l'utente fa parte
<b>-n</b>	Stampa user name e/o group name al posto degli id (da usare in combinazione con -u,-g e/o -G)

Quando invocato senza parametri, id stampa tutte le informazioni disponibili sull'utente.

# find

**find** rappresenta un'altra utility molto utile per l'amministrazione di sistema. Permette di cercare file all'interno del file system che soddisfano i requisiti richiesti dall'utente, ed eventualmente di manipolarli.

Esempio: stampa tutti i nomi delle sottodirectory contenute in \$HOME/code

```
$ find code -type d
code
code/reti_roda_webserver
code/lezione_java_thread
code/lezione_java_thread/crezione_thread
code/lezione_java_thread/cc_risolto
code/lezione_java_thread/multithreaded_server
```

# find

Esempio: trova tutti i file \*.bkp nella directory mydir ed eliminali

```
$ find mydir -name "*.bkp" -type f -delete
```

Esempio: trova tutti i file \*.c nella directory mioprogetto e contane il numero di righe:


```
$ find mioprogetto -name "*.c" -type f -exec wc -l {} \;
```

find è una utility piuttosto ostica ma estremamente potente, con cui però vale assolutamente la pena di familiarizzare

# SCP

**scp** è un comando che ci permette di effettuare una copia sicura (attraverso SSH) di file tra due host in rete. Utilizza la stessa autenticazione di ssh, stabilendo una sessione di login: dobbiamo specificare un *username* e una *password*.

Sintassi:



```
$ scp user@host:/path/to/file /destination/path
```

The diagram shows two labels with arrows pointing to the command syntax. The label "host remoto" has an arrow pointing to "host" in "user@host". The label "host locale" has an arrow pointing to the first "/" in "/destination/path".

Dove `/path/to/file` rappresenta il path al file da copiare sulla macchina `host`, mentre `/destination/path` è il file sulla macchina locale: directory corrente (`.`) o altro path sul sistema.

## SCP

Possiamo ovviamente copiare anche un file locale su un host remoto:

```
$ scp path/to/file user@host:/destination/path
```

Con scp posso copiare anche un'intera directory, utilizzando l'opzione -r:

```
$ scp -r /path/to/dir user@host:/path/to/dir
```

Altre opzioni utili sono -v per visualizzare i dettagli di trasferimento, -C per utilizzare la compressione durante la copia e -P per specificare una connection port.

# Altre utilities

xargs	Utility per la creazione di righe di comando a partire dall'output di altri programmi
pv	Utility per monitorare il progresso dei dati attraverso una pipe
xdotool	Utility per lo scripting del mouse in ambiente XWindows
xclip	Utility per la gestione della clipboard dell'ambiente XWindows
iotop, iostat	Utility per il monitoraggio del sistema di I/O
vmstat	Utility per il monitoraggio della memoria virtuale
mc	Potentissimo file manager che opera in modalità testuale
crontab	Per automatizzare l'esecuzione di comandi tramite cron



# **GESTIONE DEI PACCHETTI**

# Installazione di programmi

Il parco di pacchetti software disponibili per le principali distribuzioni ha caratteristiche di completezza, quantità e qualità.

In genere distribuzione adotta un proprio gestore:

- **apt** per distro Debian-based (Debian, Ubuntu, Linux Mint, Kali Linux, ...)
- zypper per OpenSuse
- yum per Fedora
- pacman per Arch)

Inoltre ogni package manager utilizza pacchetti di un certo formato: deb per Debian based, rpm per OpenSuse e Fedora, xz per Arch, etc...

# APT: the Advanced Package Tool

apt è uno dei package manager più utilizzato, consente di installare/rimuovere pacchetti e anche (a volte) aggiornare il sistema

Le informazioni relative ai mirror dei repository sono disponibili nel file */etc/apt/sources.list*

La prima operazione è sempre quella di fare un refresh dei repository, ossia aggiornare la lista dei pacchetti...

```
$ sudo apt update
```

# APT: the Advanced Package Tool

Una volta dato il comando di update è possibile installare un determinato pacchetto

```
$ sudo apt install <nome_pacchetto>
```

Il processo di install richiederà delle conferme all'utente (opzione `-y`).

Oppure aggiornare i pacchetti installati sul sistema:

```
$ sudo apt upgrade
```

In alternativa, è possibile automatizzare l'operazione di update e upgrade

```
# apt update && apt upgrade -y
```

# APT: the Advanced Package Tool

Modificando il file `/etc/apt/sources.list` è possibile inserire altre sorgenti di pacchetti, il formato da seguire è quello specificato nel file, esempio:

```
# General format: type uri distribution [ components ]  
deb http://archive.ubuntu.com/ubuntu xenial main restricted
```

A ogni modifica di `/etc/apt/sources.list` deve essere eseguita una operazione di update.

Inoltre è possibile automatizzare l'operazione di update

```
# apt update
```

# NFS

# Network File System (NFS)

NFS permette di condividere file system o parti di file system tra macchine distinte (introdotto da SUN 1984). NFS introduce una trasparenza per gli utenti per quel che riguarda **l'allocazione file** e la **comunicazione tra le varie macchine**.

Esempio: si vuole far vedere una parte del file system di chronos (server) a una macchina mule (client).

- Il superutente del server (chronos) deve autorizzare l'esportazione (/etc/exports) del file system richiesto
- il superutente del client (mule) deve montare (mount) il file system di chronos  
mount chronos:/usr/pippo /usr1/pluto
- A questo punto, tutta la gerarchia /usr/pippo è visibile dal file system come /usr1/pluto

# Network File System (NFS)

La prima versione del 1984, viene poi aggiornata nel corso degli anni. Attualmente le versioni più utilizzate sono la v3.+ e la v4.+.

v1+ e v2+ limitazioni dovute all'utilizzo di 32-bit, file di dimensione massima di 2GB.

La specifica di NFS v4.2 risale a fine 2016, porta molti aggiornamenti come l'utilizzo di un'unica porta per semplificare la configurazione di firewall. E' possibile configurare anche complessi identity mapping.

NFS è uno strumento utilissimo per condividere un file system in ambiente UNIX e nel mondo Kubernetes.

Una soluzione simile è rappresentata da Samba.



# Network File System (NFS)

Possibilità di montare file system nella fase di boot, configurazione di /etc/fstab (consultare il man):

```
/dev/sd0a / 4.2 rw 0 0
/dev/sd0d /usr 4.2 rw 0 0
.....
.....
chronos:/home /utenti nfs rw,... 0 0
```

Eventuali file presenti nel punto di mount (es. in /utenti) non saranno visibili

# Network File System (NFS)

## LATO CLIENT

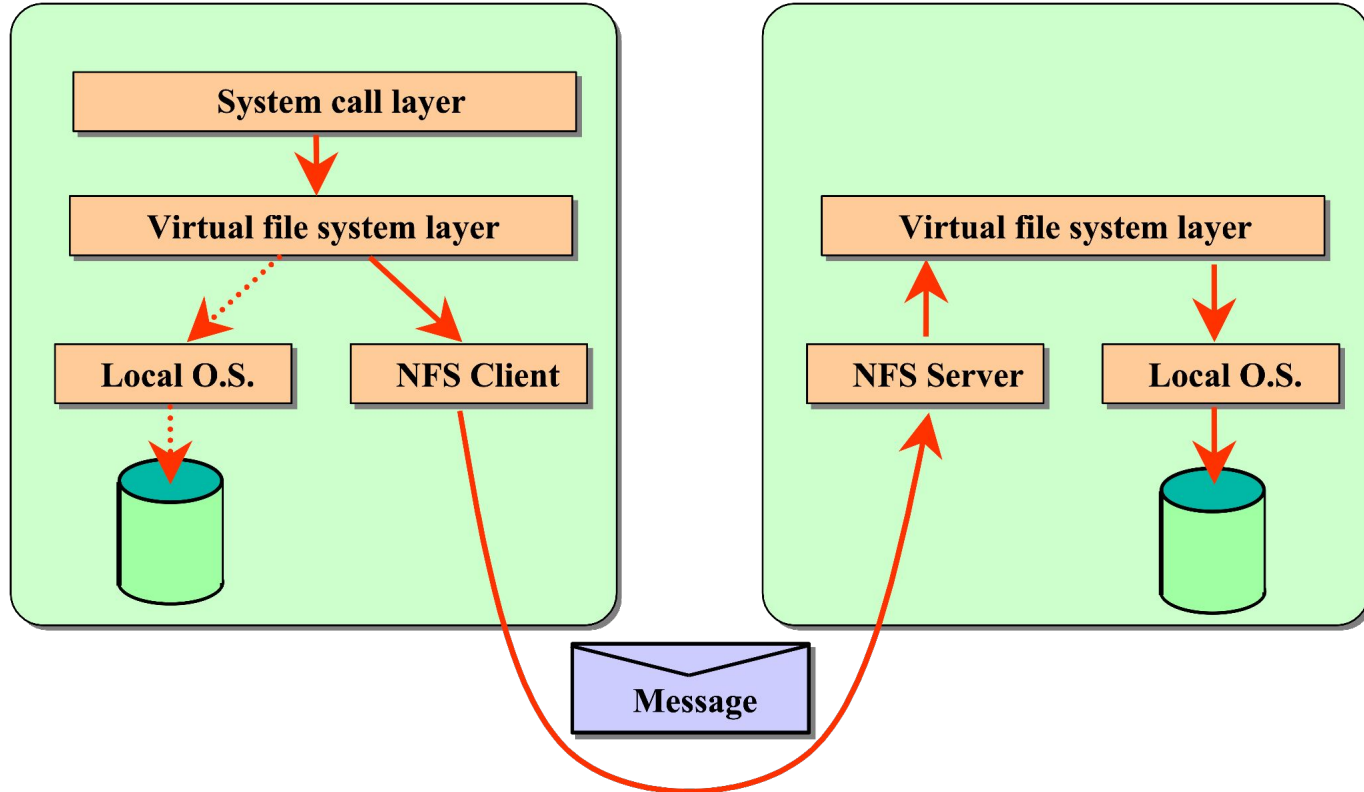
- discovery del filesystem sulla macchina server (showmount -e <ip\_address>)
- mount del file system residente sulla macchina server
- uso di tecniche di **caching** per migliorare le prestazioni (es. il demone biod)

## LATO SERVER composto di due demoni **nfsd**

- attende le richieste di mount (e controlla l'autorizzazione all'esportazione su /etc/exports)
- serve i file ai clienti (uso di più copie di nfsd per migliorare le prestazioni)

ATTENZIONE: il protocollo NFS è stateless e i comandi possono essere differenti a seconda delle distribuzioni di Linux / UNIX

# Implementazione NFS



# NFS Server-side

Per una lista delle opzioni di exports possiamo consultare il manuale (**man exports**). Vediamo un esempio di un file di /etc/exports:

```
/mnt/myshared 192.168.1.71(rw,sync)
```

rende disponibile per il mount remoto /mnt/myshared alla macchina con IP 192.168.1.71, **rw** specifica che il filesystem può essere montato con permessi di scrittura/lettura.

Possibilità di specificare un range di IP, gruppi di rete (comando **netgroup**) per definire chi può montare la directory.

Una volta modificato il file va lanciato il comando **exportfs -arv** per sync.

# NFS Client-side

Per avere una lista degli exportable filesystem possiamo lanciare il comando (dal client);

```
showmount -e <host>
```

Dopodichè possiamo utilizza il comando **mount** per fare il mounting del filesystem remoto:

```
sudo mount -t nfs -o rw 192.168.1.74:/mnt/myshared  
/private/nfs
```

Che può smontato con il comando umount:

```
umount -f -l /private/ns
```

# Network File System (NFS)

- Un utente sulla macchina client NFS deve avere un login sulla macchina server per idmapping. Sono disponibili diverse soluzioni come utente nobody (unsecure) e utilizzo di Kerberos (secure).
- NFS utilizzava storicamente il protocollo UDP. Le versioni più recenti sono configurabili (etc/nfs.conf) ma tendono a utilizzare il protocollo TCP.
- Comandi per il controllo del funzionamento di NFS: **nfsstat**
- Attenzione ai problemi di coerenza dei dati (NFS è stateless)
- Automounting, possibili replicazioni (fault tolerance) maggiore efficienza

# Network File System (NFS) - Esperienza

- Installare e configurare il servizio di NFS su una VM
- Per ubuntu i package sono (nfs-kernel-server, nfs-common)
- Abilitare con systemd i servizi per nfs
- Consultare il manuale di exports per esporre una directory al mount remoto
- Con un'altra macchina controlliamo la visibilità del mount remoto
- Proviamo a montare il file system remoto

# **CONFIGURAZIONE DEL SOTTOSISTEMA DI RETE**



# Configurazione del sottosistema di rete

- Come avete visto nel corso di Architettura di Reti, i sistemi UNIX / Linux mettono a disposizione delle utility per configurare le interfacce di rete
- Se all'inizio i sistemi Linux erano allineati alle utility standard UNIX (ifconfig, route e netstat), la situazione è leggermente cambiata con l'introduzione del pacchetto **iproute2** che introduce il comando **ip**
- Su alcune distribuzioni è rimasta una retrocompatibilità con i comandi UNIX, mentre altre richiedono l'utilizzo esclusivo di iproute2
- Il comando **ip** permette di configurare le interfacce di rete sulla totalità delle distribuzioni Linux recenti
- Esistono anche tool/configurazione di rete che sono diversi da distribuzione a distribuzione: **/etc/network/interfaces** o **netplan** in Debian, etc...

## ip

*ip* è l'utility che permette la gestione dell'intero sistema di networking di Linux

*ip* ha numerosi sottocomandi:

ip link	(dis)abilitazione di interfacce di rete
ip addr	configurazione indirizzi di rete
ip route	configurazione routing
ip neigh	configurazione ARP in IPv4 o neighbor in IPv6
ip tunnel	configurazione tunnel (es. IPv6 in IPv4)
ecc...	

NOTA: Per una configurazione più semplice si utilizzano tool di configurazione come NetworkManager. Esistono anche tool di configurazione (semi)automatici espressamente pensati per l'uso a riga di comando come netctl.

## Esempio ip addr

```
$ ip addr
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
    inet 127.0.0.2/8 brd 127.255.255.255 scope host secondary lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:1d:09:58:ad:1a brd ff:ff:ff:ff:ff:ff
    inet 10.16.2.31/16 brd 10.16.255.255 scope global eth0
    inet6 2001:760:2e01:10:21d:9ff:fe58:ad1a/64 scope global dynamic
        valid_lft 2591979sec preferred_lft 604779sec
    inet6 fe80::21d:9ff:fe58:ad1a/64 scope link
        valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:1d:e0:72:f4:93 brd ff:ff:ff:ff:ff:ff
    inet 10.14.222.142/16 brd 10.14.255.255 scope global wlan0
    inet6 fe80::21d:e0ff:fe72:f493/64 scope link
        valid_lft forever preferred_lft forever
```

# Esempi ip

Un esempio di utilizzo del comando ip con il sottocomando addr è quello di assegnare un indirizzo ip a una determinata interfaccia di rete:

```
$ ip addr add 192.168.1.136/24 dev eth1
```

Assegna l'indirizzo ip 192.168.1.136 con subnet 255.255.255.0 all'interfaccia di rete **eth1**.

Si noti che un'interfaccia di rete può essere associata a più indirizzi IP...

# Esempi ip

Con il sottocomando `route` è possibile visualizzare le route impostate e modificarle.

```
$ ip route
10.16.0.0/16 dev eth0  proto kernel  scope link  src 10.16.2.31
metric 1
10.14.0.0/16 dev wlan0  proto kernel  scope link  src
10.14.222.142  metric 2
127.0.0.0/8 dev lo  scope link
default via 10.16.0.1 dev eth0  proto static
```

**Assegnare una nuova route di default:**

```
$ ip route add default via 192.168.1.254
```

## Esempio ip route

```
$ ip route
```

```
default via 10.16.0.1 dev eth0  proto static  
10.16.0.0/16 dev eth0  proto kernel  scope link  src 10.16.2.31  
metric 1  
10.14.0.0/16 dev wlan0  proto kernel  scope link  src  
10.14.222.142  metric 2  
127.0.0.0/8 dev lo  scope link
```

```
$ ip route add 192.168.100.0/24 dev eth0 # to eth0
```

```
$ ip route add default via 192.168.1.254 # default via gw
```

## netstat

*netstat* è un'utility che permette di visualizzare informazioni sul sistema di networking, come routing table, statistiche, e applicazioni di rete attualmente in esecuzione.

netstat -s	visualizza statistiche
netstat -r	visualizza tabella di routing
netstat -i	visualizza informazioni sulle interfacce di rete
netstat -l	mostra applicazioni in ascolto su una porta (UDP o TCP)
ecc...	

# Esempio netstat -l

```
$ netstat -l -n
```

```
Active Internet connections (only servers)
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	0.0.0.0:111	0.0.0.0:*	LISTEN
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN
tcp	0	0	127.0.0.1:25	0.0.0.0:*	LISTEN
tcp	0	0	:::1:8095	:::*	LISTEN
tcp	0	0	:::111	:::*	LISTEN
tcp	0	0	:::22	:::*	LISTEN
tcp	0	0	:::1:631	:::*	LISTEN
tcp	0	0	:::1:25	:::*	LISTEN
udp	0	0	0.0.0.0:5353	0.0.0.0:*	
udp	0	0	0.0.0.0:68	0.0.0.0:*	
udp	0	0	0.0.0.0:68	0.0.0.0:*	
udp	0	0	0.0.0.0:111	0.0.0.0:*	
udp	0	0	0.0.0.0:631	0.0.0.0:*	
udp	0	0	0.0.0.0:717	0.0.0.0:*	
udp	0	0	0.0.0.0:39823	0.0.0.0:*	
udp	0	0	:::111	:::*	
udp	0	0	:::717	:::*	

```
Active UNIX domain sockets (only servers)
```

```
...
```