Breve Corso di Recupero

Appendice B

B.1. Interprete dei comandi (Shell) e comandi di base

Nell'ambiente Unix, l'amministratore prima o poi deve inevitabilmente confrontarsi con la riga di comando; ad esempio se il sistema non si avvia correttamente o consente solo l'accesso alla `rescue mode`. Di conseguenza saper gestire questa interfaccia rientra nelle competenze basilari necessarie per far fronte alle suddette esternalità.

BREVE ACCENNO Come avviare un interprete dei comandi	Potrete avviare un interprete di comandi anche durante una sessione desktop con interfaccia grafica attraverso un'applicazione denominata "terminal". In GNOME potrete raggiungerla attraverso "Activities" (a sua volta accessibile spostando il mouse nell'angolo in alto a sinistra dello schermo) digitando le prime lettere del nome dell'applicazione. In Plasma, potrete raggiungerla attraverso K → Applications → System menu.

Questo capitolo tratta i comandi di base accennandoli brevemente. Pertanto vi suggeriamo di consultare le manual pages di questi comandi per prendere visione delle loro altre numerose opzioni disponibili.

B.1.1. Come consultare la struttura ad albero delle directories (Directory Tree) e gestire i files

Se inizierete una sessione, il comando `pwd` (acronimo di `print working directory`) vi indicherà la vostra posizione corrente nel filesystem. Il comando `cd directory` (acronimo di `change directory`) viene utilizzato per navigare dalla current directory o working directory (cartella corrente) ad un'altra. La parent directory (cartella genitore o padre) viene richiesta attraverso due punti `..` diversamente attraverso un singolo punto `.` viene richiesta la current directory. Il comando `ls` (`to list`) viene utilizzato per elencare il contenuto di una directory; se non specificherete dei parametri, il comando `ls` agirà sulla current directory.

```
$ pwd
/home/rhertzog
$ cd Desktop
$ pwd
/home/rhertzog/Desktop
$ cd .
$ pwd
/home/rhertzog/Desktop
$ cd ..
$ pwd
/home/rhertzog
$ ls
Desktop Downloads Pictures Templates
Documents Music Public Videos
```

Potrete creare una nuova directory attraverso il comando `mkdir directory` , mentre potrete rimuovere un' empty directory (cartella vuota) attraverso il comando `rmdir directory` . Il comando `mv` viene utilizzato per rinominare e/o spostare files e directories, invece il comando `rm file` cancella il file specificatogli.

```
$ mkdir test
$ ls
Desktop Downloads Pictures Templates Videos
Documents Music Public test
$ mv test new
$ ls
Desktop Downloads new Public Videos
Documents Music Pictures Templates
$ rmdir new
$ ls
Desktop Downloads Pictures Templates Videos
Documents Music Public
```

B.1.2. Visualizzazione e modifica di un file di testo

Il comando `cat file` (il cui significato estensivo è "concatena i files allo standard output device") consente la lettura e la visualizzazione del contenuto di un file da terminale. Se il file è troppo grande per adattarsi alle dimensioni dello schermo, i comandi `less` o `more` vi consentiranno la lettura pagina per pagina.
Il comando `editor` avvia un editor di testo (come ad esempio `vi` o `nano`) che vi consentirà di creare, modificare e leggere files di testo. Potrete creare dei files direttamente dall'interprete dei comandi semplicemente usufruendo del redirect [la redirezione può essere messa in atto sullo standard input/output/error o per concatenare azioni input/output]: `echo "text"> file` crea un file denominato `file` il cui contenuto è "`text`". Potrete aggiungere una linea di testo alla fine del testo contenuto da questo file con `echo "testo aggiuntivo" >> file`. Fate attenzione, in quest'ultimo esempio è stato utilizzato il simbolo >> .

B.1.3. Come cercare un file e come consultare il contenuto di un file

Il comando `find directory criteria` consulta la gerarchia dei files della `directory` specificata dall'utente in base ai `criteria` (trad. in ital. "criteri") posti dallo stesso utente. L'opzione `-name` *nome* è il criterio di ricerca più comune e consente di cercare un file in base al suo nome.
Il comando `grep expression files` analizza i contenuti dei files ed estrae dal contenuto dei files le righe corrispondenti all'`espressione regolare` (andate a leggere la casella di testo "`Espressione regolare`" a pagina 283). L'opzione `-r` esegue una ricerca ricorsiva su tutti i files contenuti nella directory specificata nel parametro. Ciò semplifica l'identificazione di un file di cui si conosce una parte del contenuto.

B.1.4. Gestione dei Processi

Il comando `ps aux` elenca i processi in esecuzione e il loro identificatore `pid` (`process id`). Dopodiché attraverso il comando `kill -signal pid` potrete inviare un segnale al processo specificato nello stesso comando (a patto che l'account utente che state utilizzando abbia la titolarità dei diritti nei confronti del processo in questione). Esistono diversi segnali, fra cui i più comuni sono: il segnale `TERM` che richiede la chiusura [di un processo]; il segnale `KILL` che forza la chiusura [di un processo].
L'interprete dei comandi consente di eseguire programmi in background se aggiungerete una "*&*" ["ampersand" in ingl. o "E commerciale" in ital.] alla fine del comando. In questo modo, l'utente potrà riprendere immediatamente il controllo della shell, sebbene il suddetto comando sia ancora in esecuzione (celato all'utente ossia come processo in background). Il comando `jobs` elenca quali processi sono in esecuzione in background. Il comando `fg %job-number` (dal termine ingl. `foreground`, in ital. `primo piano`) riporta il processo in primo piano. Qualora si desideri interrompere temporaneamente (mettere in pausa) un comando in foreground (ovvero un comando che è stato avviato normalmente o che in precedenza era in background e poi è stato riportato in primo piano) dovrete utilizzare la combinazione di tasti `Control + Z`, che vi restituirà il controllo della shell. Per riavviare il processo in pausa e porlo in background, dovrete eseguire `bg %job-number` (dal termine ingl. `background`, in ital. `secondo piano`).

B.1.5. Informazioni di Sistema (System Information o System Profiler): Memoria, Spazio sul Disco, Identità

Il comando `free` visualizza le informazioni sull'utilizzo della memoria [RAM e Swap]; il comando `df` (acronimo di `disk free`) mostra lo spazio disponibile sui vari dischi [e partizioni] mounted ed accessibili nel filesystem. L'opzione `-h` di `df` (acronimo di `human readable`) consente la visualizzazione delle dimensioni dello spazio disponibile dei suddetti dischi in un'unità di misura più intuitiva per l'utente (di solito in megabyte o in gigabyte). Per le stesse ragioni, il comando `free` dispone delle opzioni `-m` o `-g` per visualizzare le informazioni rispettivamente in megabyte o gigabyte.

```
$ free
              total        used        free      shared buff/cache   available
Mem:       16279260     5910248      523432      871036     9845580     9128964
Swap:      16601084      240640    16360444
$ df
Filesystem                   1K-blocks      Used Available Use% Mounted on
udev                           8108516         0   8108516   0% /dev
tmfs                           1627928    161800   1466128  10% /run
/dev/mapper/vg_main-root     466644576 451332520  12919912  98% /
tmpfs                          8139628    146796   7992832   2% /dev/shm
tmpfs                             5120         4      5116   1% /run/lock
tmpfs                          8139628         0   8139628   0% /sys/fs/cgroup
/dev/sda1                       523248      1676    521572   1% /boot/efi
tmpfs                          1627924        88   1627836   1% /home/user/1000
```

Il comando `id` mostra l'identità dell'utente che sta eseguendo la sessione ed una lista dei gruppi di cui è membro. A volte ciò vi sarà necessario per essere in grado di verificare se siete un membro di un dato gruppo e di conseguenza se possedete la titolarità dei diritti per accedere a determinati files o devices.

```
$ id
uid=1000(rhertzog) gid=1000(rhertzog) groups=1000(rhertzog),24(cdrom),25(floppy),27(
-> sudo),29(audio),30(dip),44(video),46(plugdev),108(netdev),109(bluetooth),115(
->scanner)
```

B.2. Organizzazione della Gerarchia del Filesystem

B.2.1. La Root Directory

Un sistema operativo Debian si basa sul `Filesystem Hierarchy Standard` (`FHS`). Questo modello definisce lo scopo di ciascuna directory. Ad esempio, le top-level directories vengono delineate nel seguente modo:

- `/bin/`: programmi di base;
- `/boot/`: il kernel Linux ed altri files necessari per il processo di avvio iniziale (`early boot process`);
- `/dev/`: device files [`dispositivi a blocchi` o `files di dispositivo`, ad esempio `/dev/sda`];
- `/etc/`: configuration files [`files di configurazione`];
- `/home/`: files personali degli utenti;
- `/lib/`: librerie di base;
- `/media/*`: `mount points` [punti di montaggio] per periferiche rimovibili (CD-Rom, chiavette USB, ecc.);
- `/mnt/`: `temporary mount point` (punto di montaggio temporaneo);
- `/opt/`: applicazioni aggiuntive rilasciate da terze parti;
- `/root/`: files personali dell'amministratore (o dell'utente root);

• /run/: volatile runtime data - dati che non permangono [in gergo informatico “non persistono”] dopo un riavvio ;
• /sbin/: system programs (software di sistema) [files per l’amministrazione e la manutenzione del sistema e di conseguenza eseguibili solo dall’amministratore del sistema];
• /srv/: dati specifici dei servizi ospitati dal sistema;
• /tmp/: file temporanei, questa directory viene spesso svuotata all'avvio;
• /usr/: applicazioni condivise - questa directory possiede a sua volta delle subdirectories denominate bin, sbin, lib che seguono la stessa logica delle directories omonime - tra queste troviamo: /usr/share/ che contiene architecture-independent data; /usr/local/ consente all'amministratore di installare manualmente alcune applicazioni senza dover sovrascrivere i files gestiti dal packaging system (dpkg);
• /var/: variable data [dati contenenti valori suscettibili di modifica nel corso dell’esecuzione di un programma] dei daemons, tra cui log files [in ital. files di registro], queues [in ital. code], spools [in ital. memoria tampone o intermediaria o di transito], caches, ecc.
• /proc/ e /sys/ sono specifici del kernel Linux pur non facendo parte del modello standard FHS. Queste directories sono utilizzate dal kernel per esportare i dati nell’user space (per maggiori informazioni leggete i paragrafi B.3.4, "L’User Space" a pagina 482 e B.5 "L’User Space" a pagina 485).

Occorre precisare che diverse distribuzioni moderne, fra cui Debian, utilizzano /bin, /sbin e /lib come symlinks (collegamenti simbolici) alle correlate directories incluse in /usr, in modo da rendere disponibili in un’unica gerarchia files ad albero tutti i programmi e le librerie. In questo modo viene garantita l’integrità dei files di sistema, nonché la loro condivisione fra diversi containers.

B.2.2. La Home Directory di ciascun utente

Il contenuto della Home Directory di ciascun utente non è standardizzato, tuttavia esistono alcune convenzioni degne di nota. Innanzitutto alla home directory viene associato il simbolo tilde ("~"). Difatti l’interprete dei comandi traduce automaticamente il suddetto simbolo con il percorso della omonima directory correlata all’utente connesso alla sessione (ovvero /home/user/).
Tradizionalmente, i files di configurazione delle applicazioni [, che riguardano ad esempio le impostazioni preferite,] sono direttamente conservate nella home directory di ciascun utente, ma i loro nomi iniziano con un punto (pertanto l’email client mutt conserva i suoi files di configurazione come ~/.muttrc). Si precisa che i files con un filename che inizia con un punto vengono nascosti per impostazione predefinita; di conseguenza il comando ls richiede l'opzione -a per elencarli, mentre i graphical file managers necessitano che la visualizzazione dei files nascosti venga abilitata nelle loro impostazioni.
Diversi programmi creano una loro directory (ad esempio ~/.ssh/) quando hanno più files di configurazione da memorizzare. Alcune di queste applicazioni (come ad esempio il browser web Firefox) utilizzano la loro directory come cache per i dati scaricati. Per tale ragione le directories in questione possono occupare una notevole quantità di spazio sul disco.
L’utilizzo eccessivo di questo tipo di files di configurazione (denominati dotfiles) è stata per lungo tempo la causa principale che ha comportato il sovraccarico della home directory di ciascun utente. Fortunatamente, grazie all’impegno collettivo sostenuto dal progetto FreeDesktop.org, è stata concretizzata una nuova convenzione, denominata “XDG Base Directory Specification”, per standardizzare l'organizzazione di questi files e directories. Questo standard impone che: i files di configurazione devono essere archiviati in ~/.config; i cache files devono essere memorizzati in ~/.cache; i data files delle applicazioni devono essere salvati in ~/.local (o nelle correlate subdirectories). La summenzionata convenzione sta iniziando ad essere riconosciuta tanto che diverse applicazioni (in particolare quelle ad interfaccia grafica) hanno iniziato a conformarsi.

I Graphical Desktops normalmente mostrano il contenuto della directory `~/Desktop/` (questa cartella può avere un altro nome e di conseguenza un percorso differente in quanto il termine inglese `Desktop` può essere stato tradotto in base alla localizzazione del sistema operativo) sul desktop (ossia nella schermata rimanente dopo che tutte le applicazioni sono state chiuse o ridotte ad icone). Infine, i programmi che gestiscono le emails a volte conservano la posta in entrata nella directory `~/Mail/`.

B.3. Come funziona un computer: i diversi Layers (livelli) coinvolti

Il computer è spesso considerato come un oggetto astratto [non lett. “distante dalla nostra portata, incomprensibile ed inaccettabile”] e la sua interfaccia visibile non rispecchia la sua reale ed interna complessità. Tale complessità è in parte dovuta al numero degli elementi coinvolti; tuttavia, questi elementi possono essere distinti in Layers (livelli) sovrapposti che comunicano fra loro rispettando l’ordine in cui sono disposti ossia interagendo rispettivamente ed in base alla necessità con il livello superiore o sottostante.
L’utente finale non necessariamente se ne rende conto e fino a quando tutto funziona non ne ha l’esigenza. Purtroppo possono insorgere delle anomalie come ad esempio "la connessione Internet non funzionante" ed è fondamentale essere in grado di identificare il layer coinvolto. In questo caso le domande da porsi sono: la scheda di rete (l'hardware) funziona? viene riconosciuta dal computer? il kernel Linux la riconosce? i parametri di rete sono stati configurati correttamente? Queste sono le domande idonee da porsi per identificare il layer responsabile ed inquadrare la fonte delle problematiche.

IN PRATICA Come verificare che l'hardware funzioni	Non è sempre semplice verificare se un elemento dell’hardware funziona correttamente. Tuttavia, se non funziona del tutto, diventa estremamente facile individuarlo! Un disco rigido è costituito da uno o più piatti rotanti e da due testine mobili e magnetiche per piatto. Quando il disco rigido è acceso, emette un ronzio caratteristico dovuto alla rotazione dei piatti. Inoltre, l'energia dissipata provoca il riscaldamento del disco. Pertanto un'unità alimentata fresca e silenziosa è probabile che non funzioni. Le schede di rete includono spesso dei LEDs che indicano lo stato della connessione. Se una scheda di rete è collegata attraverso un cavo ad un hub di una rete attiva o in uno switch, almeno uno dei LEDs della scheda di rete si accenderà. Se nessuno dei LEDs della scheda di rete si accende uno dei suddetti elementi hardware è guasto, quindi dovrete testare individualmente: la scheda di rete, il dispositivo di rete connesso ed il cavo utilizzato per il collegamento. Alcune schede di espansione, in particolare le schede 3D video, hanno dei sistemi di raffreddamento integrati come dissipatori e/o ventole. Se le ventole della scheda di espansione non girano correttamente quando è attiva, si surriscalderà con conseguenti anomalie e danneggiamenti. Purtroppo questi inconvenienti si presentano anche con i processori installati sulla scheda madre ed i loro correlati sistemi di raffreddamento.

B.3.1. Il Deepest Layer, il livello più basso: l'hardware

Innanzitutto ricordatevi che un computer è un insieme di elementi hardware. Solitamente un computer è composto da una scheda madre (in ingl. denominata `mobo`, `main board`, `motherboard`) alla quale sono collegati uno o più processori, RAM, device controllers, extension slots per schede di espansione (per altri device controllers). I controllers più notevoli sono provvisti di ingressi IDE (Parallel ATA), SCSI e Serial ATA; tali ingressi sono utilizzati per collegare i dispositivi di archiviazione come i dischi rigidi. Esistono anche altri tipi di controllers con ingressi USB, che consentono di collegare un'ampia varietà di hardwares (da una webcam al termometro, dalla tastiera all'unità di controllo per la domotica) e con ingressi IEEE 1394 (Firewire). I controllers spesso consentono di collegare diversi dispositivi contemporaneamente, di conseguenza il sottosistema gestito dal controller viene denominato con il termine inglese "`bus`".

Le schede di espansione possono essere: schede grafiche (a cui è possibile collegare uno schermo), schede audio, schede di rete, ecc. Alcune schede madri includono già le summenzionate funzionalità e pertanto non necessitano di schede di espansione.

B.3.2. Lo starter: BIOS o UEFI

Gli elementi hardware non sono in grado di svolgere autonomamente le attività per cui sono stati designati senza l'ausilio di un software che li istruisca. Analogamente il sistema operativo e le applicazioni designati al controllo e all'interazione con l'hardware non possono svolgere le loro funzioni senza un computer che li esegua.
Ma anche la simbiosi fra hardware e software purtroppo non è immediata. Affinché possa avere luogo durante l'accensione del computer è necessario un'`initial setup` [genericamente un'applicazione basilare con una configurazione essenziale]. Il BIOS (o nei sistemi più moderni l'UEFI), il programma software integrato nella scheda madre del computer, svolge tale funzione e viene eseguito automaticamente all'accensione del computer. Il compito principale del BIOS (o dell'UEFI) è di "consegnare" il controllo del computer ad un software previo controllo dei requisiti minimi. Il BIOS effettuerà il suo controllo sul primo disco rigido che contiene un boot sector (denominato `MBR` acronimo di `Master Boot Record`), per poi caricare se ritenuto compatibile il contenuto del boot sector ed eseguirlo. Dopodiché, il BIOS generalmente non svolge più nessun compito fino al successivo avvio. L'UEFI invece esegue la scansione dei dischi per trovare la partizione EFI dedicata contenente le applicazioni EFI da eseguire.

STRUMENTI
TOOLS

Setup, il tool di configurazione BIOS/UEFI

Il BIOS o UEFI contiene anche una componente software denominata `Setup`, che viene utilizzata per configurare alcuni aspetti del computer. In particolare, con questo tool potrete scegliere il dispositivo da cui eseguire il boot (ad esempio, un supporto di memorizzazione USB o il lettore CD-Rom), impostare l'orologio della macchina, ecc. Per avviare questo strumento dovrete tenere premuto un tasto subito dopo aver acceso il computer. Generalmente i tasti che sono in grado di avviare Setup sono Del, Esc e più raramente F2 o F10. Il più delle volte sarà la vostra stessa macchina ad indicarveli durante il booting con una scritta che dura pochi attimi sullo schermo.

Il boot sector (o la partizione EFI) contiene a sua volta un altro software denominato boot loader, il cui compito è quello di trovare un sistema operativo ed eseguirlo. Il boot loader, non essendo installato direttamente nella scheda madre ma caricato su un disco rigido (il che lo rende più `smart`), si può gestire meglio rispetto al BIOS [non lett. "la gestione diretta del boot loader né consente una configurazione specifica per l'OS e di conseguenza più intelligente o `smart`"] (ed è per questo motivo che il BIOS non carica autonomamente il sistema operativo). Il boot loader (di solito GRUB con i sistemi Linux) elenca i sistemi operativi disponibili sulla macchina e consente all'utente di sceglierne uno da avviare. Usualmente è possibile: impostare un intervallo di tempo limite per l'utente per poter eseguire la sua scelta; configurare quale sistema operativo avviare se il suddetto tempo limite scade o qualora l'utente non effettui nessuna scelta. Inoltre è possibile definire dei parametri da trasmettere al kernel. Viene quindi cercato un kernel da avviare, caricato in memoria ed eseguito.

NOTA

UEFI, una versione moderna del BIOS

La maggior parte dei computer odierni integra sia l'avvio UEFI, sia l'avvio BIOS per garantire la retrocompatibilità con i sistemi operativi non compatibili con l'UEFI. L'UEFI ha superato diversi limiti del BIOS: grazie ad una partizione dedicata, il boot loader non deve adattarsi all'esiguo settore di avvio del `master boot record` per poi cercare il kernel da avviare. Inoltre l'UEFI senza l'ausilio di un boot loader è in grado di avviare direttamente un kernel Linux dedicato. L'UEFI è anche indispensabile per la compatibilità con il software che necessità della convalida `Secure Boot`, un tipo di tecnologia che se abilitata nell'impostazioni dell'UEFI Setup eseguirà solo software con provenienza garantita dal venditore del sistema operativo.

Il BIOS (o l'UEFI) si occupa dell'inizializzazione e del rilevamento di diversi dispositivi. Tra questi troverete naturalmente le periferiche IDE/SATA (dischi rigidi e CD-Rom/DVD-Rom), ma spesso anche periferiche PCI. I dispositivi rilevati vengono solitamente elencati nella schermata durante il boot. Se il testo scorrevole dell'elenco [in gergo tale tecnica viene definita `scrolling`] è troppo veloce sullo schermo potrete fermarlo temporaneamente premendo il tasto Pausa ed effettuare un'analisi più accurata. Non è di buono auspicio che uno dei dispositivi PCI installati non venga elencato. Nel peggiore dei casi, il dispositivo è difettoso. Nella migliore delle ipotesi funziona ma è semplicemente incompatibile con la versione del BIOS in uso oppure con la scheda madre. Purtroppo le specifiche PCI evolvono nel tempo e non è inusuale che una non recente scheda madre non supporti una moderna scheda PCI.

B.3.3. Il Kernel

L'esecuzione del BIOS/UEFI e del bootloader dura pochi secondi. Dopodiché prende il controllo una componente software del sistema operativo denominata kernel. Come un direttore d'orchestra, il kernel dirige il coordinamento tra hardware e software. Questo ruolo coinvolge il kernel in diverse attività, tra cui: driving hardware (il controllo o la guida dell'hardware), managing processes (la gestione dei processi), la gestione degli utenti e della titolarità dei diritti e dei permessi, il file system, ecc. Quindi il kernel è la piattaforma fondamentale che funge da base comune per tutti gli altri programmi del sistema.

B.3.4. L'User Space

Sebbene per convenzione tutto quello che non riguarda il kernel viene sintetizzato con il termine inglese User Space [in ital. spazio utente], il software viene comunque suddiviso in layers (livelli). Tuttavia le interazioni fra i layers sono complesse e la differenziazione non è semplice. Un programma di solito fa uso delle librerie che a loro volta invocano il kernel, ma il flusso di comunicazioni potrebbe coinvolgere anche altri programmi o altre librerie che individualmente potrebbero chiamarsi a vicenda.

B.4. Alcune funzioni svolte dal Kernel

B.4.1. Il controllo dell'Hardware

Il kernel viene utilizzato principalmente per controllare le varie componenti dell'hardware, identificarle ed effettuare il loro switch durante l'accensione del computer [non lett. il passaggio di consegne fra le componenti hardware], ecc. Inoltre il kernel rende le stesse componenti hardware accessibili al software di alto livello, con una simplified programming interface [un'interfaccia di programmazione semplificata], in modo che il software possa utilizzare le periferiche senza doversi preoccupare di dettagli di basso livello come ad esempio in quale extension slot è stata installata una scheda di espansione. L'interfaccia di programmazione supporta anche un abstraction layer [trad. lett. "livello di astrazione" - è un metodo logico che consente di celare i dettagli dei processi svolti dai sottosistemi, nonché facilita la compartizione delle attività, l'interoperabilità e l'indipendenza della piattaforma]; un esempio pratico dell'operato dell'abstraction layer può essere realizzato prendendo in considerazione l'attività svolta da un software di videoconferenza che, indipendentemente dalle caratteristiche tecniche e dalla marca, può usufruire di qualsiasi webcam; il software di videoconferenza preso in esempio non fa altro che affidarsi all'interfaccia di programmazione `Video for Linux` (`V4L`) ed il kernel a sua volta tradurrà le chiamate di funzione della suddetta interfaccia in comandi specifici per il tipo di webcam in uso.
Il kernel esporta molte informazioni sull'hardware rilevato attraverso i virtual filesystems `/proc/` e `/sys/`. Diversi tools si occupano della catalogazione di queste informazioni. Tra questi occorre citare: `lspci` (incluso nel pacchetto `pciutils`) che elenca i dispositivi PCI; `lsusb` (incluso nel pacchetto `usbutils`) che si occupa dei dispositivi USB e `lspcmcia` (incluso nel pacchetto `pcmciautils`) per le schede PCMCIA. I summenzionati tools vengono utilizzati per identificare

l'esatto modello di un dispositivo. Ciò facilita la ricerca di notizie mirate su internet ed il reperimento di documentazione pertinente.

Esempio B.1 Esempio di informazioni fornite da lspci e lsusb

```
$ lspci
[...]
00:02.1 Display controller: Intel Corporation Mobile 915GM/GMS/910GML Express
-> Graphics Controller (rev 03)
00:1c.0 PCI bridge: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) PCI Express
-> Port 1 (rev 03)
00:1d.0 USB Controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB
-> UHCI #1 (rev 03)
[...]
01:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5751 Gigabit Ethernet
-> PCI Express (rev 01)
02:03.0 Network controller: Intel Corporation PRO/Wireless 2200BG Network Connection
-> (rev 05)
$ lsusb
Bus 005 Device 004: ID 413c:a005 Dell Computer Corp.
Bus 005 Device 008: ID 413c:9001 Dell Computer Corp.
Bus 005 Device 007: ID 045e:00dd Microsoft Corp.
Bus 005 Device 006: ID 046d:c03d Logitech, Inc.
[...]
Bus 002 Device 004: ID 413c:8103 Dell Computer Corp. Wireless 350 Bluetooth
```

I programmi sopra citati hanno un opzione `-v` , che rilascia una lista di dettagli molto estesa (con informazioni a volte non necessarie). Infine, il comando `lsdev` (incluso nel pacchetto `procinfo`) elenca le diverse risorse di comunicazione utilizzate dai dispositivi.
Spesso, le applicazioni accedono ai dispositivi per mezzo dei special files creati in `/dev/` (andate a leggere al riguardo la casella di testo "Diritti di accesso ad un dispositivo" a pagina 176). Gli special files possono rappresentare dischi (ad esempio `/dev/hda` e `/dev/sdc`), partizioni (`/dev/hda1` o `/dev/sdc3`), mouses (`/dev/input/mouse0`), tastiere (`/dev/input/event0`), schede audio (`/dev/snd/*`), serial ports (`/dev/ttyS*`), ecc.

B.4.2. Filesystems

I Filesystems sono gli elementi del kernel con cui interagirete maggiormente. I sistemi Unix integrano i diversi archivi contenenti i files in una singola struttura ad albero, in modo che gli utenti (e le applicazioni) possano avere accesso ai dati semplicemente conoscendo la posizione dei files nella gerarchia.
La struttura ad albero (o gerarchia) ha inizio nella sua "radice" denominata `root` ed è contrassegnata con una barra obliqua (slash) con inclinazione a destra `/`. `root` è pertanto una directory che a sua volta può contenere delle subdirectories, ciascuna con un proprio nome identificativo. Ad esempio, la posizione della subdirectory di `/` denominata `home` è `/home/`. La subdirectory `home` può a sua volta contenere altre subdirectories e la loro posizione segue la soprastante logica. Inoltre ciascuna directory può contenere dei files e sono proprio quest'ultimi i dati effettivi. Secondo quanto sopra appena espresso il percorso `/home/rmas/Desktop/hello.txt`

rappresenta un file di testo `txt` denominato hello, archiviato nella subdirectory `Desktop` della subdirectory `rmas` della directory `home` conservata a sua volta dentro `root`. Il kernel quindi traduce il summenzionato `naming system` [sistema dei nomi] dei files nel formato di archiviazione del disco fisico in uso.
A differenza di altri sistemi, la gerarchia dei files è unica e può integrare dati da diversi dischi. Un disco viene utilizzato come `root`, mentre gli altri vengono "montati" individualmente in una directory della gerarchia ad albero (attraverso il comando Unix `mount`); con questo metodo le directories rendono i dischi montati accessibili e vengono pertanto denominate "mount points". Potrete così trasferire le home directories personali degli utenti (tradizionalmente archiviate in `/home/`) su un secondo disco rigido che, tenuti in conto gli esempi soprastanti, conterrà le directories `rhertzog` e `rmas`. Se monterete il disco in `/home/`, le home directories diventeranno accessibili attraverso i loro paths abituali, potendo usufruire al bisogno del percorso `/home/rmas/Desktop/hello.txt`.
Esistono diversi formati di filesystem, che differiscono fra loro per il metodo di archiviazione dei dati sui dischi fisici. I più noti sono `ext3` ed `ext4`, ma ce ne sono altri. Ad esempio, `vfat` è storicamente il formato utilizzato dai sistemi DOS e Windows ed è compatibile anche con Debian. In ogni caso, dovrete dapprima creare e dedicare il filesystem sul disco per poterlo poi montare ovvero dovrete effettuare una `formattazione.` Potrete mettere in atto questa procedura tramite i comandi specifici per il formato del filesystem desiderato come per esempio `mkfs.ext3` (in cui `mkfs` è l'acronimo di `MaKe FileSystem`). Questi comandi necessitano di utilizzare come parametro lo special file della partizione da formattare (ad esempio `/dev/hda1`). La procedura di formattazione è altamente distruttiva e irreversibile, a meno che non si desideri cancellare i contenuti del filesystem e procedere con nuove installazioni.
Esistono anche filesystem di rete, come ad esempio `NFS`, in cui i dati non vengono memorizzati su un hard disk locale; bensì vengono trasmessi ad un server in rete, che li memorizzerà e li ripristinerà su richiesta esplicita. Il file system abstraction esonera gli utenti da qualsiasi preoccupazione: i files rimarranno accessibili e reperibili nei percorsi standard a loro dedicati nella gerarchia.

B.4.3. Funzioni Condivise

[Genericamente una subroutine, una sequenza di istruzioni per l'esecuzione di una specifica attività, può anche essere definita: routine, subprogram (sottoprogramma), function (funzione), method (metodo), procedure (procedura). Questi termini tecnici in realtà vengono distinti fra loro dalla teoria con definizioni differenti, ma appartenendo dal punto di vista semantico allo stesso sottotipo o potendo essere inclusi in un tipo vengono associati ed utilizzati come sinonimi rispettivamente per iponimia o iperonimia].

Diverse componenti software usufruiscono delle stesse funzioni condivise pertanto la responsabilità di tali funzioni è stata affidata al kernel per centralizzazione. [In generale la centralizzazione è un processo attraverso cui le attività decisionali di un'ente vengono affidate ad un nucleo ristretto di individui - vedi anche "decisioni prese dal centro"] Un esempio dell'efficacia di tale sistema è la gestione del kernel dei filesystems, che consente ad un'applicazione di aprire con semplicità un file facendo riferimento solo al suo nome e senza dover tenere conto della posizione effettiva del file sul disco fisico. Quindi indifferentemente che il file in questione sia suddiviso in più posizioni sull'hard disk o su diversi hard disks o addirittura archiviato su un remote file server. Le funzioni condivise che riguardano la comunicazione vengono sfruttate dalle applicazioni per scambiarsi dati a prescindere dal mezzo di comunicazione [vedi modello ISO/OSI] attraverso cui avviene il trasporto dati. Difatti potrebbe trattarsi di una rete Ethernet/Wireless o del doppino telefonico.

B.4.4. Gestione dei Processi

Un processo corrisponde ad un'istanza in esecuzione di un programma. Ciò richiede della memoria per salvare sia il programma, sia i dati da questi processati. Il kernel è responsabile della creazione e del monitoraggio dei processi. Quando viene eseguito un programma, il kernel dedica della memoria al processo, carica l'`executable code` [in ital. file eseguibile o programma eseguibile o eseguibile] dal filesystem e avvia la sua esecuzione. Il kernel inoltre tiene traccia dei dettagli del processo, in particolare associandogli un numero identificativo denominato `pid` (`process identifier`).

Gli Unix-like kernels (compreso Linux), come la maggior parte dei sistemi operativi moderni, sono definiti "multi-tasking". In teoria un sistema multi-tasking può svolgere diverse attività contemporaneamente. Ma in realtà, un sistema può eseguire un processo alla volta; pertanto il kernel per poter svolgere più attività assegna ai processi frazioni di tempo strettissime [`time-slice`] ed esegue i suddetti processi a turno. Sono quindi gli intervalli di tempo (dell'ordine di un millisecondo), a dare l'illusione all'utente che i programmi vengano eseguiti simultaneamente, quando invece rimangono attivi solo durante un determinato intervallo di tempo per poi essere sospesi in attesa del loro turno. Si precisa però che il compito del kernel è di programmare le sue attività (`scheduling`) sia affinché gli utenti abbiano la suddetta illusione, sia allo scopo di massimizzare le prestazioni complessive del sistema. Pertanto gli intervalli di tempo stabiliti dal kernel non dovranno essere troppo lunghi onde evitare che l'applicazione dia l'impressione all'utente di non essere reattiva. E nemmeno troppo brevi, onde evitare che l'eccessivo cambio di stato (attiva, sospesa) delle attività possa determinare un rallentamento del sistema. Il kernel ottimizza lo scheduling attraverso le priorità assegnate ai processi. Un processo ad alta priorità verrà eseguito ad intervalli di tempo più lunghi e più frequenti rispetto a un processo a bassa priorità.

NOTA	
I sistemi multiprocessore (e le sue varianti)	Il limite dei sistemi descritto sopra dell'esecuzione di un unico processo alla volta, deve essere considerato genericamente. Difatti i sistemi `multi-core` possono eseguire un solo processo alla volta per `core` del processore. Quindi di fatto i sistemi `multiprocessore`, `multi-core` o `hyperthreading` consentono l'esecuzione simultanea di più processi. Ovviamente il metodo [o sistema `time-slicing`, `time-slice` o `quantum`] di frazionare ed assegnare a turno il tempo ad ogni singolo processo per la sua esecuzione rimane invariato, in modo che il sistema possa far fronte a diversi processi di numero maggiore rispetto a quello dei core disponibili. E quest'ultimo caso è tutt'altro che inverosimile: in quanto un basic system, anche quando non molto attivo, ha quasi sempre una decina di processi in esecuzione.

Ovviamente il kernel consente l'esecuzione indipendente di diverse istanze dello stesso programma. Ovvero a ciascuna istanza dovrà essere riservata una frazione di tempo e di memoria. In questo modo i dati di ciascuna istanza saranno indipendenti fra loro [e saranno preservati].

B.4.5. Gestione dei diritti [e dei permessi]

I sistemi Unix sono multiutente. Di conseguenza integrano un sistema che si occupa dell'assegnazione e della gestione della titolarità dei diritti degli utenti e dei gruppi di utenti; inoltre tale sistema gestisce anche le autorizzazioni, denominate in gergo informatico `permessi`, degli atti [in gergo informatico `azioni`] degli utenti e dei gruppi di utenti. Il kernel quindi gestisce per ogni processo un insieme di dati che di fatto consentono il controllo dei permessi. Di norma, i processi vengono identificati in base all'account utente che li ha messi in esecuzione. Inoltre i processi possono prendere in carico solo quelle azioni che sono state consentite dall'`owner` [`owner user` (utente titolare dei diritti) o `owner group` (gruppo titolare dei diritti) - in gergo informatico `owner` viene semplice tradotto in `proprietario`]. Ad esempio, quando viene richiesta l'apertura di un file il kernel fa un controllo incrociato sull'identità del processo e sui permessi concessi a quel processo (per maggiori dettagli fate riferimento al paragrafo 9.3 "Gestione dei diritti [permessi]" a pagina 213).

B.5. L'User Space

L'`User Space` (`Spazio Utente` in italiano) è il `runtime environment` dei processi che non sono di competenza del kernel [molto genericamente il `runtime environment` è la componente software che mette in esecuzione i processi]. Tuttavia ciò non significa che i processi dell'User Space debbano essere necessariamente stati avviati direttamente da un utente, in quanto un sistema standard esegue diversi processi denominati `daemons` (in ital. `demoni` o processi in background) a prescindere che un utente abbia avviato una sessione. I processi demoni vengono pertanto considerati di competenza dell'User Space.

B.5.1. Processi

Quando il kernel ha terminato l'inizializzazione, avvia il primo processo denominato `init` (il cui pid corrisponde a `#1`). Il processo `init` da solo non è sufficiente pertanto i sistemi Unix necessitano di ulteriori processi aggiuntivi.
Innanzitutto un processo può clonarsi (tale tecnica viene denominata `fork`). [Genericamente un `clone` è una componente hardware o software in grado di replicare tutte le funzionalità del sistema originale e di esserne indipendente]. Il kernel di conseguenza designa al processo clone una nuova area di memoria con caratteristiche identiche a quella del processo clonato. Il kernel però per distinguerli assegna al processo clone un pid differente. Per convenzione, il processo clone è denominato `child` (in gergo informatico viene tradotto non lett. "`figlio`"), mentre il processo clonato (il cui pid è rimasto invariato) è denominato `parent` (in gergo informatico viene tradotto lett. "`genitore`" e non lett. "`padre`").
In alcuni casi il processo figlio conduce la sua esistenza senza interazioni con il processo padre, disponendo dei dati duplicati del processo padre. In altri casi il processo figlio avvia un programma differente. Salvo rare eccezioni, l'area di memoria del processo figlio viene quindi semplicemente delegata a quella del nuovo programma e quest'ultimo inizia la sua esecuzione. L'`init` process (si reitera con pid `#1`) sfrutta questo meccanismo per avviare servizi aggiuntivi ed eseguire la `startup sequence` (trad. lett. sequenza di avvio). Dopodiché uno degli offpring processes [trad. lett. i processi "`progenie`" - ossia i processi avviati tramite il summenzionato meccanismo e distinti dal processo padre e dal processo figlio] lancia un'interfaccia grafica in grado di far effettuare il login agli utenti (la soprastante sequenza di eventi è descritta più dettagliatamente nel paragrafo 9.1 "System Boot" a pagina 198).
Quando un processo ultima l'attività, per la quale era stato avviato, viene `terminato`. Il kernel quindi recupera la memoria assegnatagli e non gli conferisce più frazioni di tempo (riservate all'esecuzione dei processi - `time-slicing`). Il processo padre viene istruito riguardo al termine del processo figlio: tra le altre cose, si precisa che al processo padre viene consentito di attendere il termine dell'attività delegata al processo figlio [tale sistema è definito `esternalizzazione` o `outsourcing`]. Questa procedura si manifesta palesemente nell'interprete di comando (denominato `shell`). Quando ad esempio digitate un comando in una shell, il prompt tornerà disponibile solo al termine dell'esecuzione del comando. Diversamente potrete eseguire nella maggior parte delle shells il processo in backgroud aggiungendo **&** alla fine del comando. Così facendo il prompt sarà nuovamente disponibile, ma potrebbero sorgere dei problemi qualora il comando necessiti di mostrarvi i suoi dati.

B.5.2. Demoni

Un `daemon` (`demone` in italiano) è un processo avviato automaticamente durante la `boot sequence` [genericamente l'espressione inglese `boot sequence` può riferirsi sia all'ordine di avvio dei dispositivi (`boot order`), sia alle opzioni di avvio (`boot options`)]. I demoni vengono eseguiti (in background) per operare determinate attività di manutenzione o per supportare servizi ad altri processi. Ma distinguere i demoni da altri processi solo per la caratteristica delle "attività in background" è assolutamente arbitrario in quanto non è sufficiente in riferimento all'intero sistema. In fondo i demoni sono processi come gli altri, a cui il sistema assegna frazioni di tempo per la loro esecuzione a turno con gli altri processi. Quindi è possibile distinguere i demoni dagli altri processi solo sotto l'aspetto delle interazioni con gli essere umani: difatti un processo che svolge le sue attività senza interazioni con gli utenti (ed in particolare senza interfaccia grafica) viene catalogato come "processo in backgroud" o "demone".

DIZIONARIO	
Daemon e demon hanno entrambi un significato dispregiativo?	Il termine `daemon` è un lemma inglese di origine greca a cui può essere associato anche il significato del lemma inglese `demon` (demone o presenza malvagia). Ma il significato originario del lemma daemon, in linea con la sua etimologia greca, è di spirito benevolo. Sebbene in inglese il suo significato sia sottinteso in altre lingue la sua traduzione è spesso origine di fraintendimenti.

Molti di questi demoni sono descritti dettagliatamente nel Capitolo 9, "Unix Services" a pagina 198.

B.5.3. Inter-Process Communications (IPC)

Un processo isolato, a prescindere che si tratti di daemon o di applicazioni interattive, non è molto utile, pertanto esistono diversi metodi che consentono a processi separati di comunicare tra loro o per scambiarsi dati o per controllarsi a vicenda. Genericamente la mera comunicazione fra processi viene definita `Inter-Process Communications (IPC)`.
Il mezzo più semplice per un sistema IPC sono i files. Un processo che desidera inviare dati ad un altro processo dovrà semplicemente trascriverli su un file (con un nome concordato fra i processi interessati preventivamente) in modo che possa essere aperto ed i suoi contenuti letti.
Per evitare che i dati vengano archiviati sul disco rigido, potrete utilizzare una `pipe`, un object con due capi [da intendersi come estremità]; i bytes scritti da un capo verranno letti dall'altro capo. Se i due capi sono controllati da processi distinti, si ottiene un inter-process communication semplice e con bassi costi. Esistono due tipi diversi di pipes: `named pipes` e `anonymous pipes`. Una named pipe è un entry nel filesystem (sebbene non contenga i dati da trasmettere), che entrambi i processi possono aprire autonomamente se conoscono la sua posizione. Riguardo alla comunicazione fra due processi correlati (ad esempio un parent process ed un child process), il parent process potrà creare prima del forking un anonymous pipe, che il child process erediterà. In questo modo entrambi processi saranno in grado di scambiarsi dati senza usufruire del filesystem.

IN PRATICA
Un esempio concreto

In questa casella di testo verrà descritto dettagliatamente cosa succede quando viene eseguito un complex command (una `pipeline`) in una shell. Si prenda in considerazione un processo bash (ad esempio la user shell su Debian), contrassegnata con il `pid` 4374; si consideri di aver digitato il comando `ls | sort`. La shell pertanto procede ad interpretare il comando immesso. Ciò che innanzitutto rileva sono due programmi (`ls` e `sort`), con un flusso dati che scorre da un programma all'altro (per il carattere utilizzato |, denominato `pipe`). `bash` [ossia la shell] crea una `unnamed pipe` (che inizialmente esiste solo all'interno dello stesso processo bash).
Dopodiché la shell si clona; dalla clonazione nasce un nuovo processo bash contrassegnato con il pid #4521 (i pids sono solo `abstract numbers` ovvero meri numeri [o in altre parole singole entità] non associati a nessun significato particolare) [diversamente un esempio pratico di `concrete number` è l'espressione "5 mele" dove in questo caso "5" non è una mera entità, bensì un numero associato a degli oggetti contati (le mele)]. Il processo #4521 eredita la pipe, quindi può scriverne il lato "input"; bash reindirizza anche l'output dello `standard stream` all'input della pipe [lo `standard stream` è il flusso dati dei canali standard input, output ed error che si genera fra un programma e l'ambiente che lo esegue - tipicamente ed in questo esempio la shell]. bash pertanto mette in esecuzione (e delega al suo posto) il programma `ls`, che a sua volta elencherà il contenuto della directory corrente; ls inoltre scrive sull'output dello standard stream, canale, come è stato sopra precisato, già reindirizzato e che trasmette i suoi risultati all'input della pipe.
Un'operazione simile viene eseguita per il secondo comando: bash si clona nuovamente e dalla sua clonazione nasce un nuovo processo bash contrassegnato con il pid #4522. Anche quest'ultimo, in quanto child process del processo pid #4374, eredita la pipe; bash poi collega l'input dello standard stream all'output della pipe ed esegue (e delega) il comando `sort`, il cui scopo è ordinare i dati input e visualizzarne il risultato.
Giunti a questo punto tutte le componenti coinvolte sono pronte: `ls` esplora la directory corrente ed invia l'elenco dei files alla pipe; `sort` legge l'elenco dei files e lo ordina alfabeticamente, mostrandone il risultato. I processi #4521 e #4522 vengono terminati ed il processo #4374 (che era in attesa durante l'operazione), riprende il controllo e rende nuovamente disponibile il prompt all'utente per consentirne l'immissione di un nuovo comando.

Non tutte le inter-process communications sono utilizzate dai processi per scambiarsi dati. Spesso nascono dalla necessità dei processi di scambiarsi dei `control messeges` come ad esempio "`pause execution`" o "`resume execution`" [rispettivamente per sospendere e riavviare l'esecuzione]. Unix (e quindi Linux) supporta un meccanismo denominato `signals` attraverso il quale un processo può semplicemente inviare un segnale specifico (scelto da un elenco predefinito di segnali) ad un altro processo. L'unico requisito necessario per mettere in atto il summenzionato meccanismo è il pid.
Per comunicazioni più complesse, esistono anche altri meccanismi attraverso i quali un processo può, ad esempio, consentire l'accesso o condividere una parte della sua memoria dedicata ad altri processi. La memoria condivisa può essere usata dai processi per scambiarsi dati.
Infine, persino le connessioni di rete possono contribuire alla comunicazione fra i processi; quest'ultimi possono essere eseguiti da computer differenti anche a migliaia di chilometri di distanza.
Tutti questi meccanismi sono tipici in un sistema Unix-like in modo da poter soddisfare qualsiasi esigenza.

B.5.4. Libraries (Librerie)

[La traduzione letterale e corretta in italiano dei termini `library/libraries` dovrebbe essere "biblioteca/biblioteche", ma in gergo informatico italiano viene comunemente tradotto in "libreria/librerie"]

Le `function libraries` svolgono un ruolo cruciale nel funzionamento di un sistema operativo Unix-like. Le function libraries non possono essere definite programmi, in quanto non sono in condizione di essere eseguite indipendentemente, ma raccolte di frammenti di codice utilizzati dai programmi regolari. Potrete trovare tra le librerie più comuni, ad esempio:
• la libreria `standard C (glibc)`, che contiene funzioni di base tra cui quelle necessarie per l'apertura di file o di una connessione di rete, oppure come ausilio alle interazioni con il kernel;
• `graphical toolkit`, `Gtk+` e `Qt`, che consentono a molti programmi di usufruire dei graphical objects che supportano;
• la libreria `libpng`, che carica, interpreta e salva le immagini in formato PNG.

La presenza delle librerie consente alle applicazioni di riutilizzare codice preesistente. Di conseguenza viene agevolato lo sviluppo delle applicazioni in quanto molte applicazioni sono compatibili con le stesse funzioni. Inoltre, così facendo, pur essendo le librerie spesso sviluppate da persone diverse, lo sviluppo complessivo del sistema rimane in linea con l'orientamento storico Unix.

CULTURA
Il metodo Unix: una cosa alla volta

Uno dei concetti fondamentali su cui si regge la famiglia dei sistemi operativi Unix è che ciascun tool (strumento) deve occuparsi soltanto di una cosa, ma deve farla bene; in questo modo le applicazioni possano riutilizzare i suddetti strumenti ed usufruire di una logica avanzata. Tale filosofia si manifesta in diversi modi. Un esempio classico sono gli `shell scripts` e le `complex sequence` realizzate attraverso tools basilari come: `grep`, `wc`, `sort`, `uniq`, ecc. Un'altro esempio sono le code libreries: `libpng` consente, con opzioni ed in modi diversi, esclusivamente la lettura e la scrittura di immagini in formato PNG (non fa altro); però non include funzioni per la visualizzazione o per l'editing delle immagini.

Occorre precisare che le librerie vengono spesso definite "`shared libraries`" (trad. in ital. non lett. "librerie condivise"), in quanto nonostante il kernel sia in grado di caricarle in memoria una alla volta, diversi processi possono usufruirne contemporaneamente. Ciò consente un risparmio in termini di memoria; diversamente se ipoteticamente non venisse applicato tale sistema si costringerebbe il kernel a caricare il codice delle librerie insieme ai processi interessati, tutte le volte che quest'ultimi vengono messi in esecuzione.