---

# Guida alla Comprensione dell'Installazione di Linux

Febbraio 1998

**Leo Liberti**

**Sommario** - Questa guida contiene cenni teorici e pratici che facilitano la comprensione dei vari passi dell'installazione del sistema operativo Linux. Il presente testo è basato sull'installazione della distribuzione Slackware 3.3, ma i cenni teorici sono validi per qualsiasi distribuzione o sistema operativo.

# Indice

# 1 Teoria dei Sistemi Operativi

Un **sistema operativo** è un insieme di programmi che organizzano le interazioni fra il computer e l'utente. Fra le funzioni di un sistema operativo ci sono, per esempio, la gestione della memoria, l'organizzazione del tempo utile di calcolo della CPU, la gestione di periferiche come dischi fissi e stampanti, la gestione delle connessioni di rete.

Molto in generale i sistemi operativi sono divisi in tre parti essenziali:

- Interfaccia utente, o **shell**. Può essere a riga di comando, ovvero di

testo (in questo tipo di interfaccia l'interazione con il computer avviene attraverso sequenze di comandi digitate alla tastiera) oppure a finestre, in modo grafico (in questo tipo di interfaccia si può interagire col computer per mezzo di oggetti grafici disegnati sullo schermo, come pulsanti, barre di scorrimento, liste, dialog boxes, eccetera, usando soprattutto il mouse). Esempi: il programma contenuto nel file eseguibile `COMMAND.COM` nell'MS-DOS, `bash` su Linux sono interfacce di testo; Windows 95 e X-Windows sono interfacce grafiche.

- Nocciolo del sistema, o **kernel**. È il centro del sistema operativo. Il kernel contiene le parti essenziali del sistema, come il programma che implementa la condivisione della CPU o quello che si occupa della gestione della memoria.

- Gestori di periferiche, o **device drivers**. Ogni device driver è un programma che conosce l'intimo funzionamento della periferica di cui si occupa, e permette al sistema operativo di interfacciarsi con essa senza occuparsi dei dettagli tecnici.

## 1.1 Un po' di storia.

I computer sono un prodotto della seconda guerra mondiale. Gli alleati decrittavano i messaggi radio tedeschi (cifrati secondo l'allora indecifrabile codice Enigma) servendosi di macchinari meccanici detti "bombe" per il rumore infernale che facevano. Le bombe erano state inventate e usate da John von Neumann e Alan Turing, due fra i più geniali studiosi di intelligenza artificiale di metà secolo. Alla fine della guerra molti di coloro che avevano lavorato alle bombe si misero a progettare altri macchinari per fare i calcoli, ma basate su principi elettrici anziché meccanici, e fu così che nacquero i computer. I primi computer erano immensi, occupavano interi laboratori, e disponevano di quantità di memoria nell'ordine di grandezza di 16 kilobytes. Queste macchine non avevano il sistema operativo: l'utente disponeva di una serie di interruttori che controllavano lo stato delle celle di memoria (ogni cella può "ricordarsi" solo una cosa: se passa o non passa corrente. Al passaggio di corrente si associa il numero 1, mentre se la corrente non c'è vi si associa il numero 0. Il sistema numerico così costruito viene chiamato "sistema numerico binario" o "in base due"), immetteva il programma direttamente in linguaggio macchina, bit[1] per bit, lo faceva girare e successivamente recuperava i risultati in qualche modo. Il grosso svantaggio di questo approccio è che l'utente impiegava moltissimo tempo a immettere il programma e leggere i risultati, mentre il tempo di esecuzione della CPU

[1] "Bit" sta per BInary digiT, cifra binaria. Ogni cifra binaria può essere 0 o 1.

era piuttosto corto rispetto ai tempi di attesa: in pratica la CPU (che è la risorsa più preziosa del computer) lavorava in modo inefficiente. I sistemi operativi sono nati, in origine, per sopperire a questi svantaggi, creando delle tecniche che organizzassero l'esecuzione di più programmi contemporaneamente (multiprogramming, multitasking, multithreading) e potessero gestire l'uso del sistema da più persone contemporaneamente (sistemi multiutente).

### 1.1.1 Unix

Il sistema operativo Unix è stato creato negli anni '70 nei laboratori della compagnia telefonica Americana Bell. Il nome "Unix" è stato dato per contrapporlo ad un primo tentativo fallito di sistema operativo sviluppato ai Bell Labs chiamato "Multics". Il vantaggio più immediato di Unix è la sua portatilità, cioè il fatto che esistano delle versioni di Unix quasi per ogni computer esistente al mondo. La portatilità di Unix è stata ottenuta creando un linguaggio di programmazione ad hoc: il linguaggio C, con il quale Unix va di pari passo. Ogni sistema Unix dispone di un compilatore C di serie, e un programma scritto in C per Unix dovrebbe poter essere compilato praticamente senza modifiche su ogni piattaforma Unix. Oggi con il termine "Unix" non si intende più un particolare sistema operativo, bensì qualsiasi sistema operativo conforme a certi standard. I sistemi Unix vengono usati soprattutto per computer che debbano essere affidabili, come server in rete o macchine dedicate al calcolo numerico.

### 1.1.2 DOS

Il DOS (Disk Operating System) è stato sviluppato dalla Microsoft per i primi personal computer costruiti dalla IBM nel 1979. Disegnato per poter funzionare sui primi modelli basati sul processore Intel 8088 che spesso disponevano di appena 64 kilobyte di memoria RAM e un lettore/scrittore (drive) per dischetti da 180 kilobyte, è sempre stato estremamente piccolo ed essenziale. Nessuna delle versioni del DOS ha mai incluso facilitazioni multiutente o multitasking: il DOS è stato progettato per eseguire un programma per volta. La naturale evoluzione del DOS è stato il sistema a finestre Windows 3.0, che per la prima volta portava il concetto di multitasking nei personal computer, poi evolutosi in Windows 95 e infine Windows NT, che tenta di imporsi come alternativa a Unix.

## 1.2 Processi

Un **processo** è un programma in esecuzione. Quando il sistema operativo esegue un programma per prima cosa carica il file binario che contiene il programma (cioè il file il cui contenuto è il codice in linguaggio macchina che rappresenta il programma) da una memoria di massa[2] alla memoria RAM principale del computer. Contemporaneamente al caricamento viene riservata una zona di memoria per la *tavola del processo*. La tavola del processo contiene informazioni relative al processo, come le zone di memoria occupate, le risorse che il processo deve usare, lo stato della CPU, eccetera. I dati vengono poi passati al processore che li esegue, un'istruzione alla volta, fino all'istruzione che segnala la fine del processo. A questo punto le aree di memoria occupate vengono liberate e le risorse occupate vengono sbloccate.

## 1.3 Compiti del kernel

Il kernel è, come abbiamo già detto, il nucleo del sistema operativo. Esso svolge funzioni fondamentali come la gestione del tempo processore e della memoria RAM. Vediamo più in dettaglio queste funzioni.

1. Gestione della memoria RAM.
2. Code, *time-sharing* e *scheduling*.
3. Smistamento segnali.
4. Coordinamento processi.

### 1.3.1 Gestione della memoria RAM

Il sistema operativo deve garantire una serie di funzionalità minime di gestione di memoria. Queste sono:

- Protezione. Due o più processi non possono sovrapporsi usando le stesse aree di memoria. È necessario che ogni processo usi esclusivamente la memoria che gli è stata assegnata e che nessun processo abbia la libertà di danneggiare gli altri processi.

[2]Con il termine "memoria di massa" si intende un tipo di memoria non volatile e relativamente economica. Esempi di memoria di massa sono i dischi fissi, i floppy disk, i cd-rom, i nastri.

- Trasparenza. I programmi devono ignorare l'effettivo indirizzo di memoria[3] usato. Solo il sistema operativo può decidere quale indirizzi usare.

- Codice condiviso. Se ci sono tre programmi in esecuzione e ognuno di essi deve essere in grado di stampare una stringa sullo schermo, è perfettamente inutile e molto inefficiente che ognuno di essi contenga il codice necessario a svolgere tale attività. Sarebbe molto meglio se ci fosse una sola copia di questo codice in memoria e tutti i programmi la leggessero all'occorrenza. Il sistema deve perciò essere in grado di gestire le *librerie dinamiche*.

- Memoria virtuale (*swapping*). Se la memoria RAM viene riempita il sistema deve essere in grado di liberarne delle parti scaricando dalla memoria RAM su una memoria di massa i dati usati meno frequentemente.

### 1.3.2 Code, time-sharing e scheduling

Se più processi devono accedere alla stessa risorsa, possono sorgere dei problemi di condivisione. Uso di code, time-sharing e scheduling sono tecniche per risolvere il problema. Un insieme di processi viene eseguito in coda se prima di iniziare un processo o l'uso di una risorsa si attende il termine del processo precedente. Le tecniche di time-sharing e scheduling prevedono che il tempo utile del processore e delle risorse venga condiviso da più processi contemporaneamente.

Le code di processi vengono utilizzate nei **batch systems**, cioè sistemi a lotti, dove ogni processo deve attendere il completamento del processo precedente. In questi sistemi l'utente non usa il computer in modo interattivo, bensì progetta e disegna un programma e lo fa eseguire dal sistema, che poi rende i dati in uscita. È un tipo di uso del computer molto simile al modo in cui venivano utilizzati i primi computer costruiti. I tipi di code sono fondamentalmente due:

- FIFO, o *first in first out*, cioè "primo a entrare primo a uscire". Descrive una coda in cui i processi vengono eseguiti nello stesso ordine in cui gli utenti li hanno mandati al computer.

[3] La memoria è come una griglia i cui nodi possono assumere i valori 0 o 1. Ogni nodo è un bit, quindi 8 nodi formano un byte. Ci si riferisce ai bytes tramite un indirizzo di memoria.

- SJF, o *shortest job first*, cioè "prima il lavoro più corto". Questa tecnica è migliore della FIFO in quanto il tempo medio di attesa di ogni processo è minore.

L'uso della condivisione del tempo avviene in tutti quei sistemi dove è importante la risposta in tempo reale del computer. La quasi totalità dei computer di oggi usa il time-sharing. Molto spesso l'organizzazione della condivisione avviene per mezzo di una o più code circolari (chiamate **round robin**), in cui ogni processo riceve una fetta di tempo processore fino al completamento dell'esecuzione del processo. Associate alle code o ai processi possono esserci delle priorità diverse. Ad esempio è importante che l'amministratore del sistema possa eseguire i suoi processi (che si suppongono vitali per il funzionamento stesso del sistema) senza attendere troppo. Anche in questi tipi di code possono valere le tecniche FIFO o SJF. Nel caso di quest'ultima il modo in cui la si implementa è il seguente: i processi ricevono una quantità di tempo processore che è inversamente proporzionale al **tempo di vita** del processo (cioè la quantità di tempo in cui il processo è stato in esecuzione), cosicché i processi più brevi ricevono più tempo processore.

### 1.3.3 Smistamento segnali

I processi possono scambiarsi segnali attraverso il nucleo. I segnali possono essere fra i più diversi, ma generalmente tutti i processi conoscono il segnale di TERM (chiusura processo effettuata dal processo stesso, e quindi in maniera "aggraziata"), HUP (*hang-up*, che serve a ri-iniziare un processo) e KILL (chiusura processo effettuata dal sistema operativo, e quindi in maniera "brutale").

### 1.3.4 Coordinamento processi

Due (o più) processi interessati ad una stessa risorsa possono essere in competizione o in cooperazione. Se due processi tentano di usare una risorsa che ammette solo un utente alla volta essi entrano in competizione per la risorsa. Esistono algoritmi specifici per risolvere i problemi di *deadlock*, cioè quando i due processi tentano di usare la risorsa esattamente nello stesso momento. Tali algoritmi sono basati sull'uso di **semafori**, cioè di particolari variabili che controllano l'accesso alle risorse. Un processo può usare una risorsa solo quando il semaforo è "verde".

Il caso di due processi in cooperazione può essere agevolmente compreso con un esempio. Esaminiamo il caso (reale) del sistema che gestisce il sistema di pronotazioni aeree internazionali. Un programma si occupa di raccogliere

tutte le prenotazioni dalle agenzie di viaggio, un altro programma si occupa invece di passare queste informazioni alle compagnie aeree competenti. La zona dove vengono depositati i dati relativi alle prenotazioni viene chiamata *buffer*. Supponiamo che in un dato momento non vi siano prenotazioni. È allora perfettamente inutile che il secondo programma cerchi di distribuire le prenotazioni inesistenti. I processi devono quindi cooperare e scambiarsi informazioni sullo stato della zona di buffer (piena o vuota).

## 1.4 Device drivers

I gestori di periferiche (device drivers) sono programmi che si occupano dell'interfacciamento con una particolare periferica (dispositivo hardware). Possono essere inclusi nel kernel (kernel monolitico) o in files a parte (kernel modulare).

### 1.4.1 IRQ e polling drivers

A seconda dell'architettura del computer le periferiche possono disporre o meno di linee privilegiate per dialogare con i relativi drivers. Le linee privilegiate sono chiamate **IRQ** ("Interrupt ReQuest lines") e sono basate sul concetto di **interrupt**. Un interrupt è semplicemente una serie di istruzioni in linguaggio macchina; quello che è particolare è il modo in cui queste vengono chiamate. Quando un device driver genera un interrupt il processo in esecuzione corrente si interrompe, viene salvato lo stato della CPU e vengono eseguite le istruzioni dell'interrupt. Al termine dell'interrupt lo stato della CPU viene ripristinato e il processo in attesa può riprendere la sua esecuzione. Se le periferiche sono abilitate a usare le IRQ la chiamata è diretta: la periferica può far sapere al device driver in modo attivo che ha bisogno dell'attenzione del processore. Il device driver a sua volta genera l'interrupt giusto e il processore esegue le istruzioni dell'interrupt.

Se invece le periferiche non possono usare le IRQ è il device driver che deve preoccuparsi di interrogarle in continuazione per sapere se l'interrupt debba essere generato o meno. Questa tecnica si chiama **polling**. Se un device driver è in modalità polling viene eseguito continuamente un ciclo vuoto che termina quando la periferica ha bisogno dell'attenzione del processore. A quel punto viene generato l'interrupt. Al termine dello stesso il ciclo riprende.

```
ripeti
        ripeti
                se periferica_richiede_CPU
```

```
                    esci
                fine_se
            in_continuazione
            genera_interrupt
    in_continuazione
```

Il grosso svantaggio di questa tecnica è l'altissimo costo in termini di istruzioni di CPU sprecate. Il ciclo deve ripetersi in continuazione semplicemente per restare in attesa. Il polling viene implementato in casi dove si sa a priori che lo stato della periferica non cambierà spesso, e quindi si può eseguire il ciclo non molto spesso. L'esempio più classico è la stampante, che è una periferica molto lenta. Sono molto pochi i device drivers di stampanti che usano le IRQ, mentre invece il polling è molto più usato[4].

I device drivers di periferiche veloci (come memorie di massa, schede video, schede audio, schede di rete, porte seriali, eccetera) sono quasi sempre basati sulle IRQ. D'altro canto le IRQ sono una risorsa limitata (l'architettura dei PC ne prevede 16) per via del fatto che assegnare una IRQ significa permettere che una periferica interrompa un qualsiasi processo in esecuzione, e non è desiderabile avere troppe possibilità di interruzione nel normale funzionamento del processore.

Ci sono casi in cui l'esecuzione del codice da parte della CPU non deve assolutamente essere interrotta; gli interrupts vengono allora temporaneamente disabilitati. Per esempio, se la CPU sta eseguendo il codice relativo ad un interrupt per la gestione del disco fisso e in quel momento il disco fisso manda un segnale IRQ per la richiesta di interrupt, avremmo una situazione in cui un interrupt è interrotto da un interrupt dello stesso tipo. Questo può portare problemi di inconsistenza e instabilità. Durante l'esecuzione di particolari interrupts, perciò, si preferisce disabilitare l'esecuzione di altri interrupts.

### 1.4.2 Porte di input/output

Nel paragrafo precedente è stato spiegato il meccanismo con cui hardware e CPU si scambiano le richieste di attenzione, ma non il modo in cui si scambiano i dati. Di solito vengono riservate delle aree di memoria a questo scopo. L'area di memoria deve essere almeno tanto estesa quanto il **bus** della periferica, cioè il numero di bytes alla volta che la periferica può mandare in uscita o ricevere in ingresso. Se una periferica è in grado di scambiare 2

[4]Il DOS, tutte le versioni di Windows, Linux usano per default dei "polling drivers" nell'uso delle stampanti. Un'eccezione notevole a questa regola è il sistema dBASE della Borland, che invece usa un printer driver basato sulle IRQ.

bytes alla volta, l'area di memoria deve essere di 2 bytes. Le aree di memoria destinate allo scambio dei dati sono dette **porte di i/o**.

## 1.5 La shell: dialogare con l'utente

Sebbene i sistemi operativi rendano i computer abbastanza indipendenti, è l'utente umano che ne fa uso, e quindi deve disporre di un modo per impartire comandi alla macchina. La shell, o "interfaccia utente" è un qualsiasi programma che traduce ordini provenienti dall'esterno in segnali comprensibili al kernel. Vi sono due tipi molto diffusi di interfaccia utente: l'interfaccia testo (altrimenti detta "prompt dei comandi") e l'interfaccia grafica, abbreviata con il termine GUI ("graphical user interface"). Sicuramente non sono le uniche interfacce utente a disposizione: si sta facendo strada proprio quest'anno, come interfaccia utente per il grande pubblico, l'interfaccia vocale; e già da parecchio tempo si usano interfacce di tutti i tipi per permettere a portatrici di handicap di comunicare (si pensi per esempio al fisico Stephen Hawking). Quasi tutti i sistemi operativi dispongono di un'interfaccia testo, e molti dispongono anche di un'interfaccia grafica.

### 1.5.1 Interfaccia testo

L'interfaccia testo, o prompt dei comandi, è un programma che usa tastiera e monitor per comunicare con l'utente. Generalmente vengono creati due flussi di dati detti **stdin** e **stdout** (cioè *standard input* e *standard output*) che sono associati per default rispettivamente alla tastiera e al monitor. L'utente impartisce gli ordini tramite tastiera e riceve i messaggi sul monitor. Gli ordini sono frasi di un linguaggio definito ad hoc (ogni shell ha un linguaggio diverso); ogni ordine viene terminato da un carattere CR ("Carriage Return", cioè INVIO), dunque il modo per "lanciare un ordine" è premere INVIO. La frase viene tradotta in termini comprensibili al kernel, che provvede ad eseguire l'ordine. Ogni messaggio mandato dal kernel all'utente viene affidato all'interfaccia testo che lo scrive sullo stdout (per default il monitor). Quando l'ordine finisce di essere eseguito la shell presenta una breve stringa all'utente (che si chiama *prompt*: di qui il nome *prompt dei comandi*) per indicare che il sistema è pronto a ricevere un altro ordine. I flussi di stdin e stdout possono tuttavia venire deviati, o *ridiretti*, per far sì che l'input di un ordine sia in realtà l'output di un altro. In questo modo si possono formare catene di ordini sequenziali e dipendenti che formano un "super-ordine".

Interfacce di testo più comuni usate in DOS:

- `COMMAND.COM`; questa è l'interfaccia di testo standard del DOS. È estremamente scarna e molto poco potente.
- `4DOS.COM`; prodotta dalla JPSoft, quest'interfaccia è paragonabile, come potenza, alle interfacce di testo di Unix, e in certi casi più veloce e più facile da configurare.

Interfacce di testo più comuni usate in Unix:

- `bash`; è la shell standard di moltissime versioni di Unix, ivi compreso Linux. È estremamente potente.
- `tcsh`; una versione più avanzata di `csh`, la shell che dovrebbe essere più intuitiva per i programmatori C. È paragonabile a `bash` come potenza.

### 1.5.2 Interfaccia grafica

L'interfaccia grafica è nata per supplire alla difficoltà concettuale dell'interfaccia testo. È basata largamente sul mouse, sulle finestre, su bottoni disegnati sullo schermo ed altri elementi grafici che si suppone siano di uso intuitivo ed immediato. Solitamente le interfacce di testo sono più potenti, anche se più difficili da usare, e richiedono meno risorse del computer per funzionare. Succede spesso che un'interfaccia grafica faccia pesanti richieste sul sistema in termini di performance, cosicché il *system overhead* (cioè la "fetta" di risorse che spetta al sistema operativo per funzionare) diventa molto alto.

L'interfaccia grafica del DOS è Microsoft Windows in tutte le sue versioni (Windows 95 e Windows NT sono in effetti dei sistemi operativi a sè stanti). L'interfaccia grafica più usata nei sistemi Unix è X-Windows.

## 1.6 Gestione delle periferiche in Unix

Il punto forte dei sistemi Unix è la loro portatilità. Uno degli standard che i sistemi Unix propongono ai programmi è il modo in cui vengono gestite le periferiche. *Ogni periferica può essere usata come se fosse un file.* In questo modo è sufficiente che il programmatore sappia come aprire, leggere, scrivere, chiudere un file (cosa abbastanza banale) per usare tutte le periferiche del sistema. Il modo in cui questo standard viene implementato è che ogni periferica è rappresentata da un file speciale. Generalmente su tutti i sistemi Unix questi files speciali si trovano nella directory `/dev`. I files speciali hanno nomi brevi ma descrittivi della periferica che rappresentano: gli hard disks, per esempio, sono `hd` seguiti da una lettera che indica il numero di hard disk

e da un numero che indica la partizione desiderata. Così il file speciale che indica la seconda partizione sul primo hard disk del sistema è `/dev/hda2`.

Le periferiche possono essere di due tipi: a carattere e a blocchi. Una periferica è a carattere se il flusso di input/output deve verificarsi in modo sequenziale, un dato dopo l'altro. Esempi tipici sono la tastiera, le porte seriali, i nastri per il backup. Una periferica a blocchi può invece accedere ai dati in qualsiasi posizione essi si trovino. Esempi tipici sono i dischi di tutti i tipi. Questa è la stessa differenza che c'è fra una musicassetta e un disco: per cercare un brano sulla musicassetta dobbiamo riavvolgere il nastro fino al brano cercato, quindi dobbiamo passare i dati uno a uno non potendo accedere direttamente al dato che ci interessa. Per cercare un brano su un disco, invece, basta spostare la puntina.

### 1.6.1 Periferiche a carattere

Organizzare i dati su una periferica a caratteri è arduo, se non impossibile. Spesso si cerca di creare delle code di dati in modo che gli utenti non debbano aspettare che la periferica si liberi e i dati inviati giungano a destinazione. Queste tecniche sono chiamate tecniche di **spooling**. In quasi tutti i sistemi operativi le stampanti dispongono di programmi detti *spoolers* che organizzano le code[5].

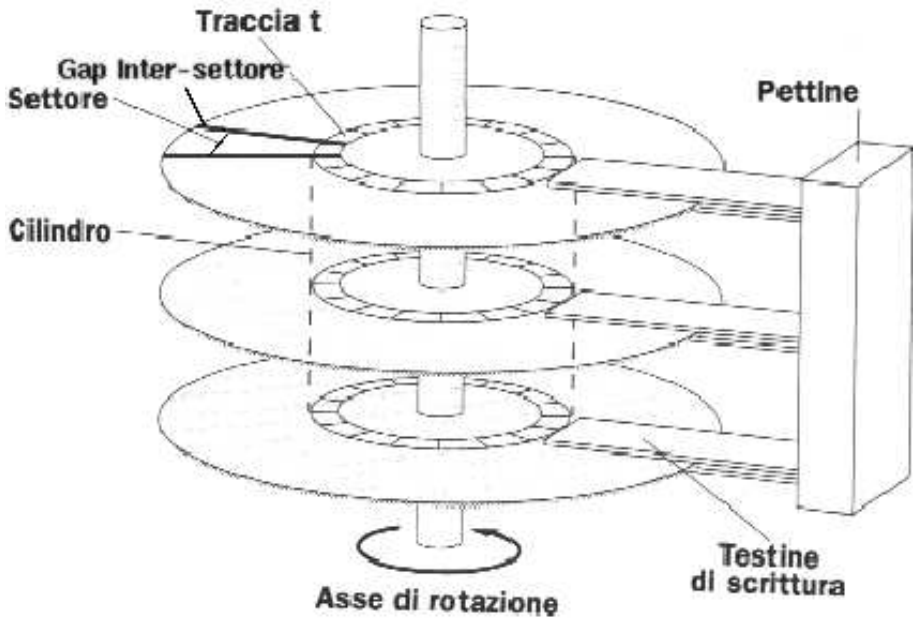
### 1.6.2 Periferiche a blocchi

Come si è detto, le periferiche a blocchi possono accedere direttamente a un qualsiasi blocco. Il blocco viene considerata l'unità minima di allocazione dati sia in lettura sia in scrittura. Il blocco è un'unità di misura legata in parte alla **geometria** del disco (cioè al tipo di hardware), in parte al tipo di **formattazione** del disco (cioè a come la superficie magnetica del disco è stata preparata per l'input/output) e in parte alla struttura del **filesystem**. Sulle periferiche a blocchi è infatti possibile montare un filesystem, cioè un'organizzazione dei blocchi in **file**. Praticamente tutti i sistemi operativi (tranne la prima versione del DOS) possono organizzare i files in un albero di **directory** e **subdirectory**. La struttura dell'albero è gerarchica, e la directory al livello più alto (sopra la quale non c'è nulla) viene chiamata directory **root**.

[5]Eccezione notevole a questa regola è il DOS.

### 1.6.3 Geometria del disco e formattazione della superficie

Un hard disk (disco rigido) è di solito composto da più piatti montati in pila che ruotano ad una velocità angolare costante. A lato del disco è montato un pettine di testine che leggono e scrivono sulla superficie magnetica delle diverse facce dei piatti.




A differenza dei dischi in vinile, su cui è inciso un unico solco (o traccia) a spirale, sui dischi magnetici vi sono numerose **tracce** concentriche. L'insieme delle tracce su tutte le facce del disco, distanti lo stesso raggio dal centro, viene detto **cilindro**. Durante la formattazione del disco (per i dischi fissi questa viene fatta dalla fabbrica produttrice, per i floppy disks è necessario effettuarla sul proprio computer) viene definito il **gap intersettoriale**: questa è una sottile striscia radiale smagnetizzata che delimita la divisione fra settore e settore (vedere figura sopra). C'è un po' di confusione sulla definizione di "settore": molto spesso con settore si intende in effetti l'intersezione fra il settore come disegnato in figura e la traccia. Il **settore** inteso in questo senso è la più piccola unità di misura di riferimento in fase di lettura/scrittura sul disco fisso[6]: si può solo leggere o scrivere un numero

[6]Può venire spontaneo chiedersi come mai un settore verso l'esterno del disco contenga tanti dati quanto un settore verso l'interno, dato che il primo è, per motivi di geometria, più lungo del secondo. La ragione è che la testina ha dei tempi minimi di lettura/scrittura della superficie magnetica, ed essendo la velocità angolare del disco una costante la velocità

di bytes che sia multiplo dei bytes contenuti in un settore (solitamente 512 o 1024 bytes). Si noti che il settore è un'unità di misura fisica, e quindi visibile solo al device driver del disco fisso, che la traduce in blocchi, l'unità minima di allocazione per il resto del sistema operativo.

Il *tempo di posizionamento testine* (*seek*) è il tempo impiegato dal pettine mobile a muovere le testine nel punto in cui deve avvenire la lettura o scrittura (nell'ordine di 10 ms). Il *tempo di latenza* è il tempo impiegato dal motore per far giungere il giusto settore nel punto in cui passano le testine (nell'ordine di 1 ms). Il *tempo medio di accesso* è definito come la media dei primi due. Il *tempo medio di trasmissione dati* è il tempo medio impiegato dai dati per arrivare dal circuito stampato dove risiede l'elettronica del disco alla scheda madre, ed è trascurabile rispetto ai primi tre.

### 1.6.4 Partizioni

In generale può essere necessario montare diversi filesystems su uno stesso disco. In questo caso si usano partizioni diverse per ogni filesystem. Una **partizione** è un'area del disco riservata ad un filesystem. Le partizioni possono essere di tre tipi: primaria, estesa e logica. Si possono montare filesystems sulle partizioni primarie e logiche. Le partizioni primarie sono partizioni da cui si può caricare il sistema operativo all'avvio della macchina (processo di boot). Le partizioni estese servono solo a contenere le partizioni logiche: sulle partizioni estese in genere non è possibile montare un filesystem. Ogni disco può contenere un massimo di 4 partizioni fra primarie ed estese e un numero imprecisato (ma comunque limitato) di partizioni logiche.

### 1.6.5 Il filesystem

Ci sono molti tipi di filesystem. Il DOS usa un filesystem basato sulla tecnica **FAT** (File Allocation Table), che registra i nomi e le posizioni dei files in alcuni blocchi riservati all'inizio del filesystem. La tecnica FAT è stata progettata in origine per uso di periferiche a blocchi non più grandi di 32 megabytes. Questa limitazione la rende estremamente inefficiente su dischi o partizioni più grandi di 32 MB. Windows 95, ancora basato sul sistema FAT anche se spesso usato con dischi di 1 gigabyte o più, costringe l'utente a utilizzare dei blocchi di 32 kb. Questo significa che ogni file, anche se piccolo, occupa 32 kb sul disco fisso. Windows NT, OS/2 e i sistemi Unix utilizzano altri tipi di filesystem che limitano il problema.

lineare risulta maggiore verso l'esterno; perchè la testina sia in grado di accedere ai dati quando la velocità lineare del disco è maggiore i dati devono essere più spaziati.

## 1.7 Multiutenza

I sistemi Unix e Windows NT sono progettati per essere usati da più persone contemporaneamente. Questo significa che devono esserci dei meccanismi che permettono al sistema operativo di riconoscere gli utenti che stanno lavorando in un dato momento e dei meccanismi per impedire agli utenti di danneggiarsi a vicenda o di danneggiare il sistema. Il sistema sa in ogni momento quali utenti si sono collegati al computer per mezzo del **processo di login**. Ogni utente che desidera connettersi deve, come prima cosa, digitare il suo nome di riconoscimento (*login*) e una parola d'ordine segreta *password*. Se l'utente è abilitato a collegarsi, cioè se possiede un **acconto** (*account*) sul computer, viene accettato, altrimenti la connessione viene chiusa. Per creare un acconto per un utente bisogna:

- dire al computer il nome di login dell'utente
- associare al nome una password
- creare una *home directory*, cioè una directory che appartiene all'utente, all'interno della quale l'utente abbia il completo controllo.

Il modo in cui viene protetta la privacy degli utenti è di assegnare dei permessi di scrittura, lettura ed esecuzione ai files sul sistema e di organizzare gli utenti in gruppi. Un file appartiene all'utente che l'ha creato, che può anche decidere di alterarne i permessi, e al gruppo a cui appartiene l'utente che l'ha creato. L'utente possessore del file può abilitare l'accesso al file in lettura, scrittura o esecuzione per sè stesso, per il gruppo a cui appartiene il file o per tutti gli utenti del sistema.

Esiste un unico utente, su tutti i sistemi, che ha il controllo totale di tutti i files e di tutte le risorse del computer. Questo utente è l'amministratore del sistema. Sui sistemi Unix l'amministratore viene chiamato **root**, sui sistemi Windows NT **administrator**. L'amministratore è (o dovrebbe essere) l'unico utente abilitato a:

- creare nuovi acconti
- montare e smontare filesystems
- spegnere e accendere il computer
- configurare programmi e files di uso comune

*Poiché l'amministratore ha il completo controllo della macchina non deve commettere errori che potrebbero danneggiare gli altri utenti.* A tal fine l'amministratore deve entrare nel sistema per il minor tempo possibile e solo quando è strettamente necessario.

## 1.8 Accensione del computer

L'insieme di eventi che accadono fra l'accensione del pulsante di accensione della macchina e il momento in cui il sistema operativo è pronto per funzionare viene chiamato **bootstrap** o **processo di boot**. Per prima cosa viene caricato in memoria un piccolissimo sistema operativo chiamato BIOS ("Basic Input Output System") che solitamente risiede o in una memoria ROM ("Read Only Memory", memoria a sola lettura) oppure in una memoria RAM che viene tenuta sempre accesa con una batteria, e che contiene alcune informazioni essenziali sulle periferiche montate sul computer. Il BIOS esegue come innanzitutto un controllo sul corretto funzionamento delle periferiche necessarie, come tastiera, monitor, disco fisso, CPU eccetera. Se il controllo dà esito negativo il computer si blocca, altrimenti passa a determinare da quale periferica eseguire il caricamento del sistema operativo vero e proprio (le periferiche più comuni usate per questo scopo sono disco fisso, floppy disk e scheda di rete). Se il sistema operativo viene caricato dalla scheda di rete il controllo passa alla memoria ROM di quest'ultima che completa il processo, altrimenti viene caricato ed eseguito il codice contenuto nel primissimo settore[7] del disco su cui si trova il sistema operativo. Questo codice si occupa di caricare in memoria il kernel del sistema operativo e di eseguirlo. Il kernel esegue poi tutte le funzioni necessarie alla configurazione e al caricamento delle altre parti del sistema. Nel caso di sistemi Unix il kernel affida la configurazione e il caricamento delle altre parti del sistema ad un programma chiamato `init`, che diventa così il "processo padre" di tutti gli altri processi che verranno eseguiti nel corso della sessione.

---

# 2 Internet

Con il termine **Internet** si intende l'insieme più grande di reti di computer collegate insieme. In questa sede ci occupiamo solo di reti locali; una rete locale è un insieme di computer collegati da uno stesso cavo. Il cavo viene

[7] Il settore in questione viene chiamato MBR ("Master Boot Record").

collegato ai computer tramite una particolare periferica chiamata **scheda di rete**. Fra i tipi maggiormente usati c'è sicuramente la scheda *Ethernet*. Per collegare due reti locali si usano dei computer detti **gateway** o **router**. Si usano i gateways quando le reti sono adiacenti: in pratica il gateway è un computer che ha due schede di rete, una collegata ad una rete e una collegata all'altra rete. Si usano i routers quando le due reti sono distanti e comunicano con una linea dedicata.

I dati vengono spezzati in **pacchetti** (*packets*) prima di essere mandati sul cavo: la ragione è che molti computer devono mandare dati sullo stesso cavo, e quindi sarebbe problematico, se non impossibile, gestire dei flussi continui di dati che avvengano contemporaneamente. L'insieme di regole con cui i pacchetti vengono mandati, controllati e ricevuti si chiama **protocollo di rete**. Esistono molti protocolli che lavorano su diversi problemi. Il modo in cui i protocolli lavorano su un pacchetto di dati è l'incapsulamento: ogni protocollo per cui passa il pacchetto aggiunge un titolo (*header*) e una coda (*trailer*) al pacchetto stesso, che via via si ingrandisce e si arricchisce di informazioni di trasmissione.

## 2.1 Protocollo Ethernet

Ogni scheda di rete Ethernet dispone di un numero unico, stampato in un circuito integrato sulla scheda, che la identifica: questo è il numero Ethernet. Il numero Ethernet è formato da una sestupla di numeri esadecimali fra 00 e FF. Un tipico numero Ethernet è 00:31:A9:F4:13:BE. Supponiamo che la scheda A debba mandare dei dati alla scheda B su una rete Ehternet. La scheda A incapsula il pacchetto di dati in un titolo e una coda in cui appare il numero della scheda A (mittente) e quello della scheda B (ricevente), poi mette il pacchetto così modificato sul cavo. Tutte le schede sulla rete leggono il titolo del pacchetto, ma solo la scheda B, il cui numero si trova nel titolo, legge il pacchetto e lo passa al sistema operativo.

Un problema che può sorgere con le reti Ethernet è che ad un certo punto due schede tentino di mettere un pacchetto sul cavo allo stesso tempo. Quando questo succede il protocollo Ethernet prevede che la prima scheda di rete che "nota" l'errore mandi un segnale di *jamming*, cioè di rete intasata. Tutte le schede Ethernet sulla rete ricevono il segnale di jamming e si bloccano per un periodo di tempo casuale, scaduto il quale la trasmissione viene riprovata[8].

[8]Sebbene in teoria sarebbe possibile che la rete possa rimanere intasata in continuazione, la probabilità che questo accada è più o meno la stessa probabilità che una persona, secondo le leggi della fisica quantistica, scompaia in un punto e ricompaia in un altro.

## 2.2 Protocollo IP

Il protocollo IP ("Internet Protocol") si occupa della trasmissione del pacchetto fuori della rete locale. A ogni scheda di rete, di qualsiasi tipo, esistente su Internet viene assegnato un numero detto "numero IP" che identifica univocamente la scheda. Poiché questo numero, a differenza del numero Ethernet, non è stampato sulla scheda, esiste un'organizzazione di controllo di Internet (`internic.net`) che assegna i numeri IP a chi ne fa richiesta.

I numeri IP sono delle quadruple di numeri fra 0 e 255. Un tipico numero IP è 130.192.238.3. I numeri IP seguono una precisa gerarchia. Per i numeri IP le reti sono divise in

- reti di classe A. I numeri di rete vanno da 0.0.0.0 a 127.0.0.0; ogni quadrupla che termina con tre zeri indica una rete, e ogni rete può avere $256^3 - 1 = 16777215$ computer. Ci sono 128 reti di classe A.
- reti di classe B. I numeri di rete vanno da 128.0.0.0 a 191.255.0.0; ogni quadrupla che termina con due zeri indica una rete, e ogni rete può avere $256^2 - 1 = 65535$ computer. Ci sono 16128 reti di classe B.
- reti di classe C. I numeri di rete vanno da 192.0.0.0 a 255.255.255.0; ogni quadrupla che termina con uno zero indica una rete, e ogni rete può avere $256 - 1 = 255$ computer. Ci sono 4128768 reti di classe C.

Si possono anche creare delle reti più piccole di 255 computer, con delle tecniche dette di *subnetting*. Per determinare la grandezza di una rete si usa un numero che si chiama **netmask**. Ogni rete ha una netmask associata. Una netmask assomiglia a un numero IP ma ha un significato diverso. Prendiamo ad esempio la netmask 255.255.255.192. Il numero di computer sulla rete è dato dalla differenza ordinata

$$\begin{aligned} 255.255.255.\ & 255\ - \\ 255.255.255.\ & 192\ = \\ 0.0.0.\ & 63 \end{aligned}$$

Quindi una netmask 255.255.255.192 corrisponde a 63 computer sulla rete.

Il numero IP viene assegnato alla scheda di rete e non al computer per tenere conto dei computer che hanno più di una scheda di rete e che quindi appartengono a più reti diverse, e che di conseguenza devono avere più numeri.

Quando un pacchetto passa per il protocollo IP viene incapsulato in un titolo e una coda che specificano il numero IP della scheda mittente e il

numero IP della scheda ricevente, oltre a varie altre informazioni riguardanti il trasferimento.

## 2.3 Protocollo TCP

Il protocollo TCP ("Transfer Control Protocol") si occupa di due cose.

1. Assicura che i dati vengano trasferiti senza errori. Il TCP pretende la sicurezza del trasferimento a scapito della velocità: è molto raro infatti che capiti di trasferire dei dati su Internet e che questi arrivino danneggiati, anche se è molto comune attendere molto tempo per il trasferimento dei dati. Ai pacchetti in partenza viene aggiunto un titolo e una coda in cui si specificano delle informazioni di controllo dei dati. All'arrivo queste informazioni vengono usate per determinare se il pacchetto è integro o no. Se il pacchetto è danneggiato viene ignorato e viene richiesta un'altra copia di quel pacchetto.

2. Divide i dati in pacchetti alla partenza e li ricompone nei dati originari all'arrivo.

## 2.4 Protocolli utente

I protocolli Ethernet, IP e TCP sono protocolli di sistema: senza di essi il trasferimento non può avvenire. Poi ci sono altri protocolli che organizzano il modo migliore di effettuare il trasferimento *avvalendosi di conoscenze a priori sui dati da spedire*, che sono detti protocolli utente. Fra questi i più usati sono: **FTP** ("File Transfer Protocol") per i trasferimenti di files generici; **SMTP** ("Simple Mail Transfer Protocol") per il trasferimento di messaggi e-mail; **NNTP** ("Network News Transfer Protocol") per i trasferimenti usati nelle USENET news; **HTTP** ("HyperText Transfer Protocol"), il protocollo usato dal sistema informatico WWW ("World Wide Web").

### 2.4.1 Networked File System

Una nota speciale, per quanto riguarda l'installazione di Linux, merita il sistema NFS ("Networked File System") implementato in rete. Tramite NFS si può accedere a un filesystem su un computer qualsiasi collegato ad Internet come se risiedesse sul computer locale, cioè il computer sul quale si sta lavorando.

## 2.5 Il numero, il nome e il dominio

Ai computer su Internet oltre ad essere assegnati dei numeri IP vengono anche assegnati dei nomi, e gli utenti di Internet quando devono riferirsi a un computer lo fanno per nome e non per numero. Come si scopre quale numero corrisponde a un dato nome? Inizialmente, quando le dimensioni di internet erano "piccole, veniva mantenuto un file di testo chiamato `HOSTS.TXT` con i nomi in ordine alfabetico e i numeri corrispondenti. Il file veniva aggiornato su un computer centrale e poi spedito ai computer che ne facevano richiesta. Oggi un approccio del genere sarebbe impensabile. Viene invece mantenuto un database distribuito in tutto il mondo su computer chiamati **nameserver**. I nameserver sono organizzati in gerarchie di dominio: ad esempio, se cerchiamo il numero IP del computer `ottobre.csea.torino.it.`, la richiesta di informazioni sul dominio `.it` viene mandata prima al nameserver di dominio globale (.). Ottenuto l'indirizzo IP del nameserver per il dominio `.it` gli si invia la richiesta di informazioni sul dominio `.torino`, e così via, fino ad arrivare all'informazione sul nome del computer (`ottobre`), richiesta al nameserver della rete in cui si trova `ottobre.csea.torino.it`.

## 2.6 Architettura Client/Server

Molto spesso c'è una netta divisione, sulle reti, fra i computer che servono come "stazioni di lavoro personali" (*workstations*) e i computer su cui risiedono le grosse quantità di informazioni (*servers*). In pratica succede che le workstations (i *clients*) fanno richieste e prelevano i dati necessari dal server. I programmi usati nelle connessioni di rete sono perciò di due tipi: i programmi client, usati sulle workstations, e i programmi che ascoltano le connessioni di rete, usati sui servers.

### 2.6.1 Demoni di rete

I **demoni** (*daemons*) sono dei programmi che ascoltano le connessioni. Quando il demone viene eseguito il processo si mette in ascolto delle connessioni (*sockets*) di rete. Quando arriva una richiesta di attenzione il processo esegue una chiamata alla funzione `fork()` e si sdoppia: in pratica viene creato un nuovo processo uguale al primo in tutto tranne che nel numero identificativo del processo (PID, "Process IDentification Number") che si occupa di seguire attivamente la richiesta di connessione, mentre il processo originale si rimette in ascolto per altre connessioni, che possono avvenire anche contemporaneamente a quella in atto. Quando una connessione finisce, il processo che se ne occupava muore. I demoni più usati sono: `in.telnetd`, per le con-

nessioni telnet; `in.ftpd`, per le connessioni FTP; `httpd`, per le connessioni HTTP; `portmap`, `rpc.nfsd` e `rpc.mountd` per il sistema NFS; `sendmail` per le connessioni SMTP.

### 2.6.2 Dalla parte del client

Il programma sul computer locale (il client) che si occupa di inoltrare una richiesta al demone presente sul computer remoto (il server) deve anche organizzare i dati in arrivo in modo che l'utente possa utilizzarli. I clients più usati sono: `telnet` per le connessioni telnet; `ftp` per le connessioni FTP; `netscape` o `lynx` per le connessioni HTTP; `mount` per il sistema NFS; `sendmail` per le connessioni SMTP.

---

# 3 Installazione di Linux

L'installazione di Linux può essere effettuata tramite risorsa locale (CD-ROM, dischetti, hard disk, eccetera) o tramite risorsa di rete (FTP o NFS). In questa sede ci occupiamo soltanto dell'installazione della distribuzione Slackware 3.3 tramite rete per mezzo del sistema NFS. La distribuzione Slackware risiede su un CD-ROM montato sul computer `w3.csea.torino.it`, numero IP 158.102.47.4, sulla directory `/cdrom/slakware`. I computer sui quali installiamo Linux sono dei 486sx 25MHz con 8 MB di memoria RAM, 80 MB di hard disk e scheda di rete NE2000. La rete a cui appartengono è 158.102.47.0 con netmask 255.255.255.192, nameserver 158.102.47.3 e router 158.102.47.60. Il dominio della rete è `csea.torino.it`. Installiamo il sistema di base, il sistema di rete e X-Windows. Prendiamo come esempio tipico di installazione il computer chiamato `settembre.csea.torino.it` con un numero IP 158.102.47.38.

## 3.1 Preparazione all'installazione

La prima cosa da fare è accertarsi di avere a disposizione tutta la documentazione necessaria e un secondo computer già funzionante per verifiche, controlli e imprevisti (che secondo la legge di Murphy si verificano sempre).

Si procede alla creazione di un disco di boot (che serve a far partire il kernel di Linux per l'installazione) e di un disco di root (che contiene un

filesystem minimo con i programmi necessari per l'installazione). Le immagini dei dischi (i files che contengono, blocco per blocco e indipendentemente dal filesystem, i dati che devono essere scritti sui suddetti dischi) si trovano su `w3.csea.torino.it` nelle directories `/cdrom/bootdsks.144` (per il disco di boot) e `/cdrom/rootdsks` (per il disco di root). È necessario scaricare i files su un computer locale e copiarli sui dischetti. Per lo scaricamento di usa un client FTP. Per prima cosa bisogna entrare sul computer locale come amministratori (**root**), poi al prompt dei comandi si digita

```
ftp w3.csea.torino.it
```

si procede al processo di login per `w3` e quando si è sul sistema si digita

```
bin
cd /cdrom/bootdsks.144
get net.i
cd /cdrom/rootdsks
get color.gz
bye
```

Una volta scaricati i files si copiano su dischetto:

```
dd if=net.i of=/dev/fd0
dd if=color.gz of=/dev/fd0
```

## 3.2 Partizioni, filesystem e scheda di rete

1. Si infila il dischetto di boot (quello su cui è stato copiato tt net.i) nel drive della macchina su cui si installa Linux (`settembre`) e si fa il reboot della macchina. Al prompt "`boot:`" si preme INVIO. Si prende nota della porta di input/output della scheda di rete e del suo IRQ. Si seguono le istruzioni sullo schermo: quando il sistema lo richiede si toglie il disco di boot e si inserisce quello di root, poi si preme INVIO. Dopo qualche minuto appare un invito al processo di login. Digitare "root" come login (amministratore) e niente password.

2. Si usa il programma `fdisk` per cancellare le partizioni esistenti e creare delle partizioni su cui installare Linux. Ci devono essere come minimo due partizioni primarie, una delle quali (`/dev/hda1`) più o meno di 8 megabytes configurata come "Linux swap" e un'altra (`/dev/hda2`) configurata come "Linux native". NOTA BENE: In realtà, se si dispone di abbastanza spazio su disco, la cosa più opportuna sarebbe creare più partizioni:

(a) partizione su cui montare il filesystem `/` (root), primaria, abbastanza piccola (50-90 MB). Su questo filesystem ci sono i file relativi all'avvio della macchina (in `/boot`), i files di configurazione (in `/etc`), i files che rappresentano i devices (in `/dev`), i files variabili (in `/var`), i files temporanei (in `/tmp`), i dati dell'amministratore (in `/root`), le librerie di base (in `/lib`) e gli eseguibili di base (in `/bin` e `sbin`).

(b) partizione swap, primaria; le dimensioni della partizione di swap dipendono dall'uso che si vuol fare del sistema. Di regola si usa una partizione swap che sia grande il doppio della memoria RAM sul sistema.

(c) partizione estesa di 300 MB o più, a seconda degli usi del sistema, su cui creare le seguenti partizioni logiche.

(d) partizione su cui montare il filesystem `/usr`, logica, di 200 MB o più. La directory `/usr` contiene tutti i files (eseguibili, librerie, documentazione) della distribuzione. Di regola una installazione completa di una distribuzione prende 300-400 MB, ma raramente si necessita di una installazione completa.

(e) partizione su cui montare il filesystem `/usr/local`, logica, di 50 MB o più. La directory `/usr/local` contiene tutti i files (eseguibili, librerie, documentazione, sorgenti) installati dall'amministratore.

(f) partizione su cui montare il filesystem `/home`, logica, di 50 MB o più. La directory `/home` contiene tutti i dati degli utenti del sistema, a parte l'amministratore. *La partizione* `/home` *è la più preziosa del sistema, perché i dati utente sono gli unici che quando vanno persi richiedono molto lavoro per essere riprodotti*[9]. *È necessario perciò tenere sempre backups aggiornati della directory* `/home`. La cosa migliore (avendone la possibilità) sarebbe montare `/home` su un hard disk diverso.

3. Si inizializza la partizione di swap:

```
mkswap /dev/hda1
swapon /dev/hda1
```

4. Si creano i filesystem:

```
mke2fs /dev/hda2
```

[9]Questo in parte vale anche per i dati contenuti in `/etc` e in `/usr/local`, quindi è consigliato tenere backups aggiornati anche di queste directories.

5. Si configura l'interfaccia (scheda) di rete:

   ```
   ifconfig eth0 158.102.47.38 netmask 255.255.255.192 up
   route add 158.102.47.4
   ```

6. Si monta la directory remota tramite NFS:

   ```
   mount 158.102.47.4:/cdrom /floppy
   ```

## 3.3 Copia dei files e prima configurazione

È ora possibile lanciare il programma di setup con

```
setup
```

Si seguono le istruzioni sullo schermo fino ad installazione avvenuta. Successivamente, sempre da `setup`, si installa LILO, il caricatore del kernel per far partire Linux direttamente da disco fisso. Parte poi automaticamente un programma per la prima configurazione. È necessario specificare tipo di mouse, nome, numero IP del computer locale, gateway e nameserver di rete, fuso orario della località.

Quando `setup` termina si smonta il disco di rete:

```
umount /floppy
```

Si fa il reboot della macchina.

## 3.4 Configurazione avanzata

Quando il sistema ritorna vivo è probabile che la scheda di rete non sia stata riconosciuta (viene chiaramente segnalato sullo schermo). Si entra come root e si cambia il file `/etc/rc.d/rc.modules`:

```
vi /etc/rc.d/rc.modules
```

Si toglie il carattere # (cancelletto) davanti alla linea

```
#/sbin/modprobe ne io=0xNNN
```

e si cambia "NNN" con il valore della porta i/o della scheda di rete segnato in precedenza, aggiungendo anche l'argomento "`irq=`" e il numero di IRQ segnato in precedenza. Assumendo che `io=0x360` e `irq=5` avremo

```
/sbin/modprobe ne io=0x360 irq=5
```

Si dà un altro reboot. Quando la macchina ritorna viva la scheda di rete dovrebbe essere abilitata.

Si cambia la password di root lanciando il programma `passwd` e seguendo le istruzioni sullo schermo. Si creano uno o più acconti per gli utenti per mezzo del comando `adduser`. L'uso di `adduser` è intuitivo: le uniche cose richieste sono il login dell'utente che si sta creando e la sua password. Per il resto è sufficiente dare INVIO come risposta ad ogni altra domanda. *Anche l'amministratore deve avere un semplice acconto utente con il quale usare il sistema quando non deve amministrarlo.*

L'installazione è terminata[10].

---

# 4 Uso del sistema

Seguono ora le descrizioni di due tipi di comandi: quelli elementari del sistema locale e i più usati clients di rete.

## 4.1 Sistema locale

I comandi fondamentali sono riassunti nella seguente tabella.

[10]Volendo è possibile cambiare il messaggio di login. Il testo del messaggio di login viene tenuto nel file `/etc/issue`, che viene cancellato e riscritto ogni volta che si accende la macchina. È perciò necessario, oltre a cambiare il contenuto di `/etc/issue`, commentare (cioè rendere inattive: basta mettere un cancelletto (#) all'inizio della riga) le righe di codice che cancellano il file ogni volta che si accende il computer. Detto codice si trova quasi alla fine del file di configurazione `/etc/rc.d/rc.S`.

| Unix | Corrispettivo DOS | Significato |
|---|---|---|
| `pwd` | `CD` | Nome directory corrente |
| `cd` *nome_directory* | `CD` *nome_directory* | Cambia directory |
| `cd` | Non applicabile | Vai alla *home* directory |
| `mkdir` | `MD` | Crea directory |
| `rmdir` | `RD` | Elimina directory se vuota |
| `cp` *file_sorg. destinaz.* | `COPY` | Copia files |
| `mv` *file_sorg. destinaz.* | `MOVE` e in parte `REN` | Muovi files |
| `rm` *nome_file* | `DEL` | Cancella files |
| `ls` | `DIR /W` | Mostra files senza dettagli |
| `ls -la` | `DIR` | Mostra files con dettagli |

Quello che segue è un estratto della "Guida dell'Utente di Linux" di Larry Greenfield nella traduzione di Eugenia Franzoni. La guida è liberamente distribuita dall'associazione Pluto (`http://www.pluto.linux.it`).

### 4.1.1 Nome della directory corrente

`pwd`

Usare le directory sarebbe scomodo se doveste scrivere il percorso completo ogni volta che voleste accedere ad una directory. Invece, le shell Unix hanno una caratteristica chiamata directory "corrente" o "di lavoro". Il vostro setup probabilmente vi mostra la directory corrente nel prompt: `/home/larry`. Se non lo fa, provate il comando `pwd`, per **p**rint **w**orking **d**irectory (visualizza la directory di lavoro). (Talvolta il prompt vi mostrerà il nome della macchina. Una cosa del genere è utile solo in un ambiente di rete con molte macchine diverse).

```
mousehouse>pwd
/home/larry
mousehouse>
```

### 4.1.2 Cambia directory

`cd` [*directory*]

Come potete vedere, `pwd` mostra la vostra directory corrente—un comando molto semplice. Molti comandi agiscono, per default, nella directory corrente. Per esempio, `ls` senza nessun parametro mostra il contenuto della directory

corrente. Possiamo cambiare la directory corrente usando il comando `cd`. Per esempio, provate:

```
/home/larry$ cd /home
/home$ ls -F
larry/     sam/       shutdown/  steve/     user1/
/home$
```

Se omettete il parametro opzionale *directory*, ritornerete alla vostra home directory, o directory di origine. Altrimenti `cd` vi porterà nella directory specificata. Per esempio:

```
/home$ cd
/home/larry$ cd /
/$ cd home
/home$ cd /usr
/usr$ cd local/bin
/usr/local/bin$
```

Come potete vedere, `cd` vi permette di usare percorsi sia assoluti sia relativi. Un percorso "assoluto" inizia con `/` e specifica tutte le directory prima della directory cercata. Un percorso relativo è in relazione alla directory corrente. Nell'esempio precedente, quando ero in `/usr`, ho fatto uno spostamento relativo a `local/bin`—`local` è una directory dentro `usr`, e `bin` è una directory dentro `local`.

Ci sono due directory usate *solamente* nei percorsi relativi: "`.`" e "`..`". La directory "`.`" si riferisce alla directory corrente e "`..`" è la directory madre, cioè quella che contiene la directory corrente. Queste sono directory "scorciatoia". Esistono in *ogni* directory, ma non seguono il concetto di "cartella in cartella". Anche la directory di root ha la parent directory—è la directory di root stessa!

Il file `./chapter-1` è il file chiamato `chapter-1` nella directory corrente. Occasionalmente è necessario inserire il "`./`" per far funzionare alcuni comandi, sebbene questo sia raro. In molti casi, `./chapter-1` e `chapter-1` sono la stessa cosa.

La directory "`..`" è molto utile per tornare indietro nell'albero:

```
/usr/local/bin$ cd ..
/usr/local$ ls -F
```

```
archives/  bin/       emacs@     etc/       ka9q/      lib/
/usr/local$ ls -F ../src
cweb/      linux/     xmris/
/usr/local$
```

In questo esempio, sono passato nella directory madre usando `cd ..` e ho fatto l'elenco della directory `/usr/src` da `/usr/local` usando `../src`. Notate che se fossi stato in `/home/larry`, inserire `ls -F ../src` non sarebbe andato bene!

La directory `~/` è equivalente alla vostra home directory:

```
/usr/local$ ls -F ~/
/usr/local$
```

Potete vedere che non c'è niente nella vostra home directory. `~/` diverrà più utile quando impareremo come manipolare i file.

### 4.1.3 Crea ed elimina directory

`mkdir` *directory1* [*directory2* ... *directoryN*]

Creare directory è estremamente semplice sotto Unix, e può essere un ottimo strumento di organizzazione. Per creare una nuova directory, usate il comando `mkdir`. `mkdir` sta per **m**a**k**e **dir**ectory (crea directory).

Vediamo un piccolo esempio per vedere come lavora questo comando:

```
/home/larry$ ls -F
/home/larry$ mkdir report-1993
/home/larry$ ls -F
report-1993/
/home/larry$ cd report-1993
/home/larry/report-1993#
```

`mkdir` può gestire più di un parametro, e ne interpreta ognuno come una directory da creare. Si possono specificare sia percorsi assoluti sia relativi; nell'esempio precedente, `report` è un percorso relativo.

```
/home/larry/report$ mkdir /home/larry/report/chap1 ~/report/chap2
/home/larry/report$ ls -F
chap1/  chap2/
/home/larry/report$
```

L'opposto di `mkdir` è `rmdir` per **re**m**o**ve **dir**ectory (rimuovi directory). `rmdir` funziona esattamente come `mkdir`.

Un esempio di `rmdir` è:

```
/home/larry/report$ rmdir chap1 chap3
rmdir: chap3: No such file or directory
/home/larry/report$ ls -F
chap2/
/home/larry/report$ cd ..
/home/larry# rmdir report
rmdir: report: Directory not empty
/home/larry$
```

Come potete vedere, `rmdir` si rifiuta di rimuovere directory che non esistono, come anche directory che contengono qualcosa. (Ricordatevi che `report` ha una subdirectory all'interno, `chap2`!) C'è un'altra cosa interessante su `rmdir`: cosa succede se tentate di rimuovere la vostra directory corrente? Proviamo:

```
/home/larry$ cd report
/home/larry/report$ ls -F
chap2/
/home/larry/report$ rmdir chap2
/home/larry/report$ rmdir .
rmdir: .: Operation not permitted
/home/larry/report$
```

Un'altra situazione che potreste considerare è cosa succede se provate a rimuovere la madre della directory corrente. In effetti questo non è un problema: la madre della directory corrente non è vuota, e quindi non può essere rimossa!

Tutte queste directory sono bellissime, ma non ci servono a niente, se non come posto dove immagazzinare i dati. Gli Dei dello Unix hanno visto questo problema, e l'hanno risolto dando agli utenti i file.

I principali comandi per manipolare i file in Unix sono `cp`, `mv` e `rm`. Rispettivamente questi stanno per **c**o**p**y (copia), **m**o**v**e (sposta) e **re**m**o**ve (rimuovi).

### 4.1.4 Copia files

`cp` [-i] *origine destinazione*
`cp` [-i] *file1 file2 . . . fileN directory di destinazione*[11]

`cp` è uno strumento molto utile sotto Unix, ed estremamente potente: permette ad una persona di copiare più informazioni in un secondo di un monaco amanuense in un anno.

Fate attenzione con `cp` se non avete molto spazio nel disco. Nessuno vuole vedere il messaggio `Error saving--disk full` mentre sta copiando dei dati importanti. `cp` può anche sovrascrivere i file esistenti senza avvisare—parlerò di questi pericoli più avanti.

Per prima cosa parleremo della prima linea del modello del comando: il primo parametro di `cp` è il file da copiare, il secondo è dove copiarlo. Si può copiare con un nuovo nome o su una diversa directory. Proviamo alcuni esempi:

```
/home/larry# ls -F /etc/passwd
/etc/passwd
/home/larry# cp /etc/passwd .
/home/larry# ls -F
passwd
/home/larry# cp passwd frog
/home/larry# ls -F
frog  passwd
/home/larry#
```

Con il primo `cp` ho copiato il file `/etc/passwd`, che contiene i nomi di tutti gli utenti del sistema Unix e le loro password (criptate), nella mia home directory. `cp` non elimina il file sorgente; in questo modo non ho fatto niente che possa danneggiare il sistema. Così adesso esistono due copie di `/etc/passwd` nel mio sistema, entrambe chiamate `passwd`, una nella directory `/etc` e una in `/home/larry`.

Poi ho creato una *terza* copia di `/etc/passwd` quando ho scritto `cp passwd frog`: abbiamo ora `/etc/passwd`, `/home/larry/passwd` e `/home/larry/frog`. Il contenuto di questi tre file è lo stesso, anche se hanno nomi diversi.

`cp` può copiare file tra directory se il primo parametro è un file e il secondo una directory. In questo caso, il nome breve del file resta lo stesso.

[11] `cp` nella sua sintassi ha due linee perché il significato del secondo parametro può essere diverso a seconda del numero di parametri.

Può copiare un file e cambiarne il nome se entrambi i parametri sono nomi di file. Questo è un pericolo di `cp`. Se io avessi scritto `cp /etc/passwd /etc/group`, `cp` avrebbe creato un nuovo file con contenuto identico a `passwd` e l'avrebbe chiamato `group`. Comunque, se `/etc/group` esisteva già, `cp` avrebbe distrutto il vecchio file senza darvi la possibilità di salvarlo! (Non avrebbe nemmeno stampato un messaggio facendovi presente che stavate distruggendo un file copiandocene un altro sopra!)

Vediamo un altro esempio di `cp`:

```
/home/larry# ls -F
frog  passwd
/home/larry# mkdir passwd_version
/home/larry# cp frog passwd passwd_version
/home/larry# ls -F
frog         passwd        passwd_version/
/home/larry# ls -F passwd_version
frog  passwd
/home/larry#
```

Come ho usato `cp`? Evidentemente `cp` accetta *più* di due parametri (è la seconda linea nel modello del comando). Quello che ha fatto il comando precedente è stato copiare tutti i file elencati (`frog` e `passwd`) e metterli nella directory `passwd_version`. In effetti `cp` può prendere un numero qualsiasi di parametri, interpretando i primi $n - 1$ parametri come file da copiare, e l'$n^{\text{mo}}$ come directory in cui copiarli.

Non si possono rinominare i file quando se ne copiano più di uno alla volta—mantengono sempre il loro nome. Questo porta ad una domanda interessante: cosa succede se scrivo `cp inittab pippo pluto`, dove `inittab` e `pippo` esistono e `pluto` non è una directory? Provate e vedrete.

### 4.1.5 Cancella files

`rm` [-i] *file1 file2 . . . fileN*

Adesso che abbiamo imparato a creare milioni di file con `cp` (e credetemi, troverete presto nuovi modi per creare altri file), è utile imparare anche come eliminarli. È molto semplice, il comando che state cercando è `rm`, e funziona proprio come state pensando: qualsiasi file che date come parametro a `rm` viene cancellato.

Per esempio:

```
/home/larry$ ls -F
inittab      pippo     rc_version/
/home/larry$ rm inittab pippo pluto
rm: pluto: No such file or directory
/home/larry$ ls -F
rc_version/
/home/larry$
```

Come potete vedere, `rm` non è molto amichevole. Non solo non chiede conferma, ma elimina file anche se l'intera riga non è corretta. Questo può essere pericoloso. Considerate la differenza tra questi due comandi:

```
/home/larry$ ls -F
pippo  pluto/
/home/larry$ ls -F pluto
pippo
/home/larry$ rm pluto/pippo
/home/larry$
```

e questo

```
/home/larry$ rm pluto pippo
rm: pluto is a directory
/home/larry$ ls -F
pluto/
/home/larry$
```

Come potete vedere, cambiare *un solo* carattere comporta un esito molto diverso del comando. È vitale controllare il comando inserito prima di battere INVIO.

### 4.1.6 Muovi files

`mv` [-i] *vecchio-nome nuovo-nome*
`mv` [-i] *file1 file2 . . . fileN nuova-directory*

Infine, l'altro comando che dovrete conoscere è `mv`. `mv` è simile a `cp`, tranne che elimina il file originale dopo averlo copiato. quindi molto simile all'uso di `cp` e `rm` insieme. Vediamo cosa possiamo fare:

```
/home/larry$ cp /etc/inittab .
/home/larry$ ls -F
inittab
/home/larry$ mv inittab pippo
/home/larry$ ls -F
pippo
/home/larry$ mkdir report
/home/larry$ mv pippo report
/home/larry$ ls -F
report/
/home/larry$ ls -F report
pippo
/home/larry$
```

Come potete vedere, `mv` rinomina un file se il secondo parametro è un file. Se il secondo parametro è una directory, `mv` sposta il file nella nuova directory mantenendo lo stesso nome breve.

Dovete fare molta attenzione con `mv`—non controlla per vedere se il file di destinazione esiste già, e rimuoverà tutti i file che si trova tra i piedi. Per esempio, se ho già un file chiamato `pippo` nella mia directory `report`, il comando `mv pippo report` eliminerà il file `~/report/pippo` e lo sostituirà con `~/pippo`.

In effetti, c'è un modo per far sì che `rm`, `cp` e `mv` chiedano conferma prima di eliminare file. Tutti e tre i comandi accettano l'opzione `-i`, che li fa chiedere conferma all'utente prima di cancellare qualsiasi file. Se usate un **alias**, potete fare in modo che la shell faccia `rm -i` automaticamente quando digitate `rm`.

## 4.2 Clients di rete

I clients di rete più usati sono riassunti nella tavola seguente.

| Comando | Significato |
|---|---|
| `telnet` *nome.computer* | Controllo remoto di un altro computer |
| `ftp` *nome.computer* | Trasferimento files |
| `pine` | Posta elettronica |
| `lynx` *URL* | USENET News e World Wide Web (solo testo) |

### 4.2.1 Telnet

`telnet` [*nome.computer.remoto*]

Il client telnet stabilisce una connessione con il demone `in.telnetd` del computer remoto. Viene presentata una richiesta di login e successivamente un prompt dei comandi, cosicché l'utente può controllare il computer `nome.computer.remoto` come se ci stesse davanti, anche se in effetti può trovarsi in qualsiasi parte del mondo. Si esce da telnet digitando `exit` al prompt dei comandi. Il carattere di interruzione di telnet è `C-]`, cioè tenere premuto `CTRL` e premere "parentesi quadra chiusa" contemporaneamente.

### 4.2.2 Ftp

`ftp` [*nome.computer.remoto*]

Il client ftp si occupa di trasferire files dal computer remoto al computer locale e viceversa. Il prompt di ftp è

```
ftp>
```

Si può richiedere una lista di comandi possibili digitando `help` al prompt. I comandi più utili sono

- `bin` – imposta trasferimento di files binari
- `asc` – imposta trasferimento di files di testo
- `ls` – mostra directory remota
- `cd` – cambia directory sul sistema remoto
- `get` *file* – trasferisce un file dal sistema remoto al sistema locale
- `put` *file* – trasferisce un file dal sistema locale al sistema remoto

### 4.2.3 Pine

`pine`

Il lettore di posta `pine` funziona in modo intuitivo e dispone di parecchia guida in linea, perciò una descrizione in questa sede è superflua. Basti sapere che organizza i messaggi di posta elettronica in cartelle; la cartella "INBOX" è la casella della posta in arrivo.

### 4.2.4 Lynx

`lynx [`*URL*`]`

"URL" significa Uniform Resource Locator e consiste di una stringa di caratteri che indica:

1. il protocollo da usare nel trasferimento. I protocolli possibili sono `http` (World Wide Web) e `news` (USENET news).
2. il computer remoto da contattare.
3. il file da leggere.

Per esempio,

```
http://www.eklektix.com/lwn
news://news.polito.it/it.comp.linux
```

Con le frecce su e giù si passa da un *link* all'altro; con la freccia a destra si segue un *link* e con la freccia sinistra si torna indietro. Con "g" si cambia URL, con "d" si scarica un *link* sul computer locale e con "q" si esce.