

# Laboratorio di Informatica di Base

## Laurea in Informatica Multimediale

Docente: *Andrea Fusiello*  
[profs.sci.univr.it/~fusiello](http://profs.sci.univr.it/~fusiello)

Lucidi a cura di  
Andrea Colombari, Carlo Drioli e Barbara Oliboni

*Lezione 2*

# Il File System

*Materiale tratto dai lucidi ufficiali a corredo del testo:*

**D. Sciuto, G. Buonanno e L. Mari**

**“Introduzione ai sistemi informatici” - 2005 – McGrawHill**

*e dal testo di riferimento*

**M. Bertacca, e A. Guidi**

**“Introduzione a Linux” - McGrawHill**



## Gli obiettivi del File System di un S.O.

- Gestire in modo efficiente la **memoria di massa**
- Presentare all'utente l'**organizzazione logica** dei dati (ad es. in file e cartelle) e le **operazioni** che è possibile compiere su di essi
- Fornire all'utente e ai programmi applicativi alcuni servizi di base:
  - La **creazione/cancellazione** di file e cartelle
  - La **manipolazione** di file e cartelle esistenti
  - La **copia** e lo **spostamento** di dati su supporti diversi
  - L'associazione tra file e dispositivi di memorizzazione secondaria (memorie di massa)
  - La gestione di **collegamenti (link o alias)** tra file e cartelle.  
Un collegamento è un riferimento ad un oggetto (file o cartella) presente nel file system.

# Il File System

## ■ I dati vengono organizzati in **file**

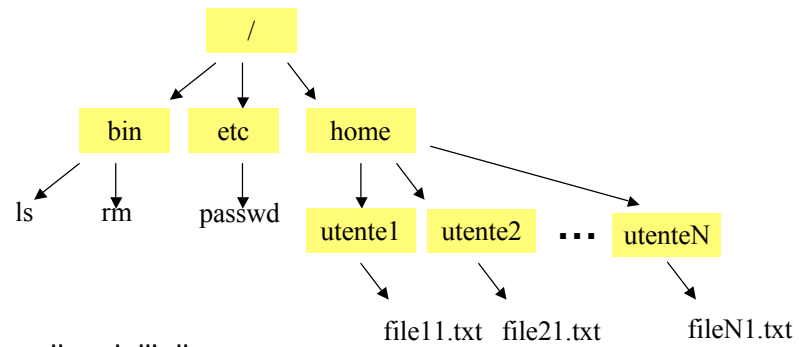
- Un file è un contenitore logico di informazioni (dati o istruzioni)
- Ogni file è identificato da un **Identificatore** o **filename** (nome.estensione), dalla **periferica** (drive) e dal **percorso** (path) sulla periferica, da varie altre informazioni (data di creazione e di ultima modifica, dimensione, diritti di accesso al contenuto del file, ecc...)
- I file possono essere raggruppati in più contenitori logici, **cartelle** o **directory**, e **sottocartelle** o **sottodirectory**, organizzati secondo una struttura gerarchica ad albero
- I **collegamenti** (o **link**, alias) permettono di creare riferimenti ad altri oggetti (file e directory) nel file system. Permettono di accedere ad un oggetto da più punti dell'albero.

# Il File System di Linux

- Opera su 5 tipi file:
  - **normali**  
Archivi di dati, testi, comandi, programmi sorgente, eseguibili.
  - **directory**  
Insiemi di sottodirectory e file normali.
  - **device**  
Dispositivi hardware collegati, vengono visti come file speciali.
  - **pipe**  
File speciali che permettono lo scambio di dati sincrono tra due processi concorrenti.
  - **link**  
Riferimento ad un altro file o directory. Le operazioni sul link si riflettono sull'oggetto collegato.

# Struttura logica

## ■ Esempio: parte di un file system



- / (root): radice dell'albero
- bin, etc, home: directory di sistema
- utente1, utente2, ..., utenteN: directory e file utente
- ls, rm, passwd: eseguibili (comandi)

## Struttura logica (2): pathnames

- Un file è individuabile attraverso il **nome** e le **sottodirectory** del percorso dalla root /

Esempio: `/home/utente1/file11.txt`

- I cammini possono essere **relativi** (rispetto a directory di lavoro) o **assoluti**

Esempio: cammino assoluto e cammino relativo rispetto alla directory utente1

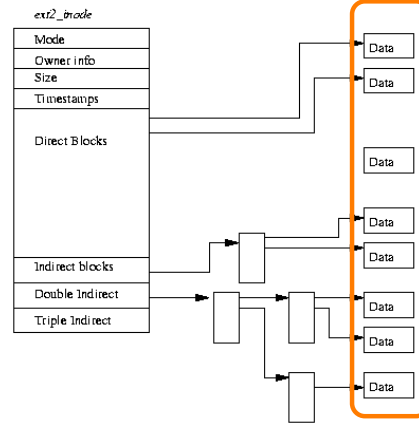
```
$ ls /home/utente1/subdir1/file1.txt
```

```
$ ls subdir1/file1.txt
```

# Organizzazione fisica

## ■ Caratteristiche del file system ext2:

- I file sono contenuti in blocchi di dati e i **blocchi** hanno tutti la stessa dimensione (tipicamente 1k, 2k o 4k)

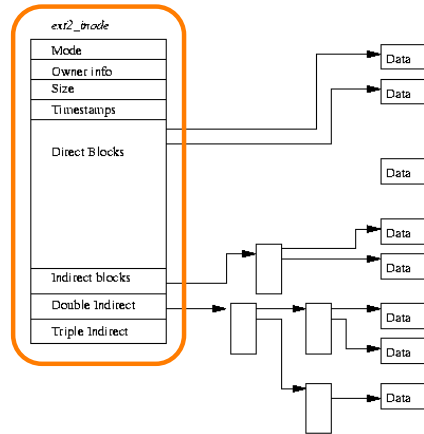




# Organizzazione fisica

## ■ Caratteristiche del file system ext2:

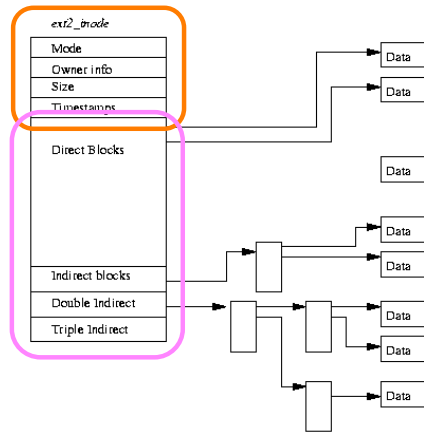
- I file sono contenuti in blocchi di dati e i **blocchi** hanno tutti la stessa dimensione (tipicamente 1k, 2k o 4k)
- Ogni file è descritto mediante una tabella (**i-node**)



# Organizzazione fisica

## ■ Caratteristiche del file system ext2:

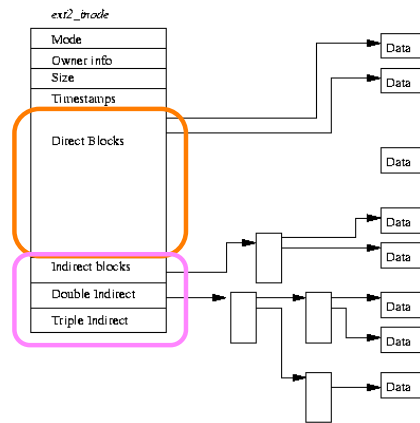
- I file sono contenuti in blocchi di dati e i **blocchi** hanno tutti la stessa dimensione (tipicamente 1k, 2k o 4k)
- Ogni file è descritto mediante una tabella (**i-node**) che contiene le **informazioni sul file** e i **riferimenti ai blocchi** del file



# Organizzazione fisica

## ■ Caratteristiche del file system ext2:

- I file sono contenuti in blocchi di dati e i **blocchi** hanno tutti la stessa dimensione (tipicamente 1k, 2k o 4k)
- Ogni file è descritto mediante una tabella (**i-node**) che contiene le informazioni sul file e i riferimenti ai blocchi del file
- I primi 13 riferimenti a blocchi sono **diretti**, i restanti sono **indiretti**



## Comandi principali (1)

- Elencare il contenuto di una cartella

```
ls [opzioni...] [cartella...]
```

- Opzioni:

- -l (informazioni estese)
- -a (visualizza file nascosti, cioè iniziati con il .)
- -R (visualizza sottocartelle)

- Esempio:

```
$ ls -laR
```

La data mostrata per default è quella di modifica. Per visualizzare la data di accesso, usare l'opzione -u

## Comandi principali (2)

### ■ Cambiare la cartella corrente

```
cd path_nuova_directory
```

#### ■ Opzioni:

- cartella corrente: .
- cartella padre: ..
- home directory: ~

### ■ Esempio:

```
$ cd ..  
$ cd ./home/mialogin/miacartella (se esiste)
```

### ■ Visualizzare path assoluto cartella corrente

```
pwd
```

## Comandi principali (3)

- Creare nuove cartelle

```
mkdir nome_cartella
```

- Esempio:

```
$ mkdir nuovacartella1 nuovacartella2
```

- Eliminare una cartella

```
rmdir nome_cartella
```

Per creare invece un file vuoto, usare il comando touch. Se il file esiste già, il comando aggiorna la data di accesso e di modifica del file.

## Copiare, spostare e cancellare i file

- Copiare file e cartelle

```
cp [opzioni...] sorgente... destinazione
```

- Spostare o rinominare file e cartelle

```
mv [opzioni...] sorgente... destinazione
```

- Eliminare file (opzione `-i` per modalità interattiva)

```
rm [opzioni...] file
```

Si consiglia la modalità interattiva: una volta cancellato un file non è più recuperabile!

## File di testo

- Per file di testo si intende un file che contiene semplicemente caratteri ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, ovvero *Codice Standard Americano per lo Scambio di Informazioni*).
- Spesso identificati dall'estensione “.txt” ma non è un obbligo. Infatti, per esempio, anche i file contenenti il codice sorgente dei programmi sono file di testo, ma assumono estensioni diverse a seconda del linguaggio di programmazione utilizzato (.c, .cpp, ecc.).



# Operazioni su file di testo

- Visualizzare uno o più files

```
cat [opzioni...] [file ...]
```

- Visualizzare file lunghi con comando di avanzamento

```
more [opzioni...] [file ...]
```

- Visualizzare file lunghi con comandi di scorrimento avanti/indietro

```
less [opzioni...] [file ...]
```

- Restituisce linee che contengono una espressione regolare in uno o più file

```
grep [opzioni...] regexp [file ...]
```

cat file1 file2 > file3 concatena I due argomenti in file3

cat senza argomenti attende da stdin

| more viene spesso usato per poter leggere l'output di comandi quando è troppo lungo

## Operazioni su file di testo

- Contare caratteri, linee e parole

```
wc [opzioni...] [file ...]
```

- Ordinamento alfabetico

```
sort [opzioni...] [file ...]
```

- Processare testo per colonne

```
cut -f list [-d delim] [-s] [file ...]
```

## Operazioni su file di testo: grep

Si considera il file “miofile”  
contenente un elenco di parole:

```
$ cat miofile
cr
ca
car
cor
caar
```

I seguenti metacaratteri possono essere usate nelle **espressioni regolari** per la ricerca di parole nel file con *grep -E*

- \* : il carattere precedente compare zero o più volte

Esempio:

```
$ grep 'ca*r' miofile
cr
car
caar
```

I metacaratteri usati nelle espressioni regolari sono simili a quelli usati nel globbing ma assumono un significato in alcuni casi diverso.

Gli apici singoli “proteggono” l’espressione evitando l’espansione da parte della shell.

# Operazioni su file di testo: grep

- `[str]` : un qualunque carattere in *str*

Esempio:

```
$ grep 'c[ao]r' miofile  
car  
cor
```

- `[a-z]` : un qualunque carattere tra a e z

Esempio:

```
$ grep 'c[m-p]r' miofile  
cor
```

- `[^str]`: un qualunque carattere **non** in *str*

Esempio:

```
$ grep 'c[^o]r' miofile  
car
```

# Operazioni su file di testo: grep

- . : un qualunque carattere

Esempio:

```
$ grep 'c.r' miofile  
car  
cor
```

- {n,m} : il carattere precedente almeno n e al più m volte

Esempio:

```
$ grep -E 'ca{1,2}r' miofile  
car  
caar
```

- ? : il carattere precedente compare zero o una volta

Esempio:

```
$ grep -E 'ca?r' miofile  
cr  
car
```

**-E indica extended regular expressions (EREs) defined in IEEE POSIX 1003.2 (Section 2.8).**

## Operazioni su file di testo: grep

- `()` : l'espressione tra parentesi viene trattata come un carattere

Esempio:

```
$ grep -E 'c(aa)*r' miofile
cr
caar
```

- `|` : compare una delle due espressioni a destra e a sinistra del simbolo (o entrambe)

Esempio:

```
$ grep -E 'ca|or' miofile
ca
car
cor
caar
```

Esempio:

```
$ grep -E 'c(a|o)r' miofile
car
cor
```

**In `$ grep -E 'ca|or' miofile` e' come fare `grep ca` poi `grep or` ed unire i risultati**

## Operazioni su file di testo: grep

- ^ : inizio riga
- \$ : fine riga
- \< : inizio parola
- \> : fine parola
  
- Condideriamo ora il seguente file di esempio

```
quale  
le stesse  
questo file  
tra le linee
```

## Operazioni su file di testo: grep

- Righe che terminano con **le**
- Righe con parole terminanti in **le**
- Righe che contengono esattamente la parola **le**
- Righe che iniziano con la parola **le**

```
$ grep -E 'le$' miofile
quale
questo file
```

```
$ grep -E 'le\>' miofile
quale
le stesse
questo file
tra le linee
```

```
$ grep -E '\<le\>' miofile
le stesse
Tra le linee
```

```
$ grep -E '^le\>' miofile
le stesse
```



## Operazioni su file di testo: grep

- Se si cerca una sequenza di caratteri in cui compare uno degli appena elencati metacaratteri (es. cerco la stringa “pippo?”), tale carattere deve essere preceduto da \ (es. cerco “pippo\?”) altrimenti sarà interpretato come metacarattere e non come semplice carattere (si dice che va “protetto”).
- La stessa cosa vale per il \ stesso, essendo anch'esso un carattere speciale.

metacarattere	tipo	significato
^	B	inizio della linea
\$	B	fine della linea
\<	B	inizio di una parola
\>	B	fine di una parola
.	B	un singolo carattere (qualsiasi)
[str]	B	un qualunque carattere in <i>str</i>
[^str]	B	un qualunque carattere non in <i>str</i>
[a-z]	B	un qualunque carattere tra a e z
\	B	inibisce l'interpretazione del metacarattere che segue
*	B	zero o più ripetizioni dell'elemento precedente
+	E	una o più ripetizioni dell'elemento precedente
?	E	zero od una ripetizione dell'elemento precedente
{j,k}	E	un numero di ripetizioni compreso tra j e k dell'elemento precedente
s t	E	l'elemento s oppure l'elemento t
(exp)	E	raggruppamento di <i>exp</i> come singolo elemento

dove B (basic) indica che la sequenza di caratteri è utilizzabile sia in `grep` che in `egrep`, mentre E (extended) indica che la sequenza di caratteri è utilizzabile solo in `egrep` (o in `grep` usando l'opzione `-E`).

# Lavorare sul File System



*Testo di riferimento:*  
**M. Bertacca, e A. Guidi**  
**“Introduzione a Linux”**  
**McGrawHill**

## Permessi e protezioni

---

- A file e cartelle sono assegnati dei **permessi** che garantiscono l'integrità e la riservatezza dei dati
- Ciascun file è collegato ad un utente, detto **proprietario**, e ad un **gruppo**
- Affinché un utente possa creare, cancellare o utilizzare un file deve possedere i permessi necessari per quella operazione

## Permessi e protezioni (2)

- I permessi si possono visualizzare con il comando `$ ls -l`

```
Carlo@your-ae2c3fc363 /cygdrive/e/Recycled
$ ls -l
total 2449
-rw-r--r-- 1 Carlo Nessuno 2474233 Jun 29 16:34 De1.zip
-rw-r--r-- 1 Carlo Nessuno    820 Jun 30 13:47 INF02
-rw-r--r-- 1 Carlo Nessuno    65 Jun 30 00:18 desktop.ini
Carlo@your-ae2c3fc363 /cygdrive/e/Recycled
$
```

## Permessi e protezioni (2)

- I permessi si possono visualizzare con il comando `$ ls -l`

```
Carlo@your-ae2c3fc363 /cygdrive/e/Recycled
$ ls -l
total 2449
-rw-r--r-- 1 Carlo Nessuno 2474233 Jun 29 16:34 De1.zip
-rw-r--r-- 1 Carlo Nessuno 820 Jun 30 13:47 INF02
-rw-r--r-- 1 Carlo Nessuno 65 Jun 30 00:18 desktop.ini
```

Carlo@your-ae2c3fc363 /cygdrive/e/Recycled  
\$

Permessi   Proprietario   Gruppo

# Codifica dei permessi

- I *permessi*:  
i primi 10 caratteri sono suddivisi in 4 campi secondo la struttura:

```
-rw-r--r-- 2 Carlo Nessuno 0 Jul 2 09:47 prova.txt
```



l u g o

- **l**: specifica il tipo di file (- = file normale; d = directory; c = file di i/o, es terminale o stampante; b = file su blocchi di caratteri, es hd; p = pipe; l = link )
  - **u**: permessi concessi al proprietario del file
  - **g**: permessi concessi ai membri del gruppo
  - **o**: permessi concessi agli altri utenti
- 
- I permessi **u**, **g**, ed **o**, sono formati da tre caratteri che specificano i permessi di lettura ( r ), scrittura ( w ) ed esecuzione ( x ).

# Codifica dei permessi per i file

- Il primo carattere di ogni insieme indica il permesso relativo alla lettura del file:
  - - la lettura non è permessa
  - **r** la lettura è permessa
- Il secondo carattere di ogni insieme indica il permesso relativo alla scrittura:
  - - la scrittura non è permessa
  - **w** la scrittura è permessa
- Il terzo carattere di ogni insieme indica il permesso relativo alla esecuzione:
  - - la esecuzione non è permessa
  - **x** la esecuzione è permessa



## Codifica dei permessi per le dir

---

- Il significato di **r**, **w**, e **x** per le directory è il seguente:
  - **r** è permesso leggere il contenuto delle directory
  - **w** è permesso modificare il contenuto delle directory
  - **x** è permesso usare pathname che contengono la directory

# Cambiare i permessi

- Cambiare il proprietario di un file o una directory

```
chown [-opzioni...] nuovo_utente file ...
```

- Cambiare il gruppo di un file o una directory

```
chgrp [-opzioni...] nuovo_gruppo file ...
```

- Cambiare i permessi di un file o una directory

```
chmod [-opzioni...] modifica_permessi file ...
```

# Cambiare i permessi: esempi

Il comando chmod permette di cambiare i permessi con

- operatore di assegnazione (=)

Esempio:

```
$ chmod u=rwx miofile
$ chmod go= miofile
$ chmod a=rx miofile
```

**NB:**  
"a" => all (tutti)

- operatori di aggiunta (+) e eliminazione (-)

Esempio:

```
$ chmod go-rx miofile
$ chmod a+rx miofile
```

- codifica numerica: "111" = "001001001" = "--x--x--x"  
"321" = "011010001" = "-rx-r---x"  
....

Esempio:

```
$ chmod 000 miofile
$ chmod 777 miofile
```

## File di tipo link

- Lo scopo dei link è potersi riferire a file e directory tramite **due** o **più pathname** (link nella home ad un file usato spesso e con path molto lungo)
- Tipi di link:
  - **hard link**: nell' i-node di un file è memorizzato il n. di riferimenti al file. Quando si aggiunge un link a quel file, il n. di riferimenti viene incrementato, e tutte le operazioni su uno dei due file si riflette anche sull'altro. Non può essere usato per le cartelle.
  - **soft link** (o **link simbolici**): file speciali che contengono un pathname. Quando in un comando si usa un link simbolico per riferirsi a un file, il sistema individua il file sostituendo il pathname nel comando.

## Creazione di link

- Sintassi del comando per un hard link

```
ln [-opzioni] nomefile nomelink
```

- Sintassi del comando per un soft link

```
ln -s [-opzioni] nomefile nomelink
```

## Visualizzazione dei link

- Con il comando `$ ls -l` vengono visualizzate informazioni sul numero di link per file e directory e sulla natura del file

```
Carlo@your-ae2c3fc363 ~  
$ ls -l  
total 2  
lrwxrwxrwx 1 Carlo Nessuno 5 Jul 2 09:44 link1 -> tempi  
-rw-r--r-- 2 Carlo Nessuno 0 Jul 2 09:47 linkhardaprova  
lrwxrwxrwx 1 Carlo Nessuno 9 Jul 2 10:03 linksoftaprova -> prova.txt  
-rw-r--r-- 2 Carlo Nessuno 0 Jul 2 09:47 prova.txt  
drwxr-xr-x+ 2 Carlo Nessuno 0 Jun 8 10:35 tempi
```

## Eliminazione di link

- Con il comando `$ rm nomelink` è possibile cancellare un link
- Nel caso di **hard link**: il comando provoca un **decremento del numero di riferimenti** nell' i-node del file collegato. Quando questo numero assume valore zero, il file è **rimosso dal disco** e l' i-node viene reso disponibile per altro utilizzo
- Nel caso di **soft link**: il comando provoca la cancellazione unicamente del pathname sostitutivo e **mai di file o directory** a cui il link si riferisce

# File di tipo device

---

- Caratteristiche dei **file device**
  - In Linux ogni entità è rappresentata sotto forma di file, comprese le periferiche (device) collegate al computer.
  - Si opera sui device con le stesse modalità con cui si opera sui file normali. Operazioni logiche di lettura e scrittura su device corrispondono fisicamente al recupero di dati dal dispositivo e all'invio di dati al dispositivo.



## File di tipo device (2)

- Tutti i file di tipo device **risiedono nella cartella /dev**
- Gli oggetti device sono caratterizzati da due numeri, **major number** (classe/tipo del device) e **minor number** (identifica un device all'interno di una classe), che identificano la periferica:

```
Carlo@your-ae2c3fc363 /home/carlo
$ ls -l /dev/lp*
crw-rw-rw- 1 Carlo Nessuno 6, 1 Jul  2 10:52 /dev/lp1
crw-rw-rw- 1 Carlo Nessuno 6, 2 Jul  2 10:53 /dev/lp2
```

Nell'esempio sopra il major number è 6, il minor number è 1 o 2 rispettivamente.

Il device speciale /dev/null elimina tutti i dati che riceve. In lettura si comporta come un file vuoto. Utile per redirectione o come argomento vuoto.

## File di tipo pipe: definizione

- Una **pipe** è un file che funziona da serbatoio FIFO. FIFO è acronimo di First In First Out, ovvero, «il primo a entrare è il primo a uscire», e a volte viene indicato con il termine coda.
- Si usano file di questo tipo per permettere a due processi di comunicare. Il primo apre il file in scrittura, e vi aggiunge dati, il secondo lo apre in lettura e lo legge sequenzialmente.
- Per creare una pipe si usa il comando:

```
mkfifo [-opzioni...] file ...
```

## File di tipo pipe: esempio

- Esempio: creando due file FIFO, si ottiene lo stesso risultato di una pipeline come **cat mio\_file | sort | lpr**

<code>\$ mkfifo fifo1 fifo2</code>	Crea due file FIFO: fifo1 e fifo2
<code>\$ cat mio_file &gt;&gt; fifo1 &amp;</code>	Invia mio_file a fifo1 in sottofondo (&)
<code>\$ sort &lt; fifo1 &gt;&gt; fifo2 &amp;</code>	Esegue il riordino di quanto ottenuto da fifo1 e invia il risultato a fifo2 in sottofondo (&)
<code>\$ lpr &lt; fifo2</code>	Accoda la stampa di quanto ottenuto da fifo2

# Stato del sistema

*Materiale tratto testo di riferimento*  
**M. Bertacca, e A. Guidi**  
**“Introduzione a Linux” - McGrawHill**



## Il SO Linux: i processi

---

- Linux è un sistema operativo multitasking: può eseguire “contemporaneamente” più programmi
- Un programma in esecuzione è definito **processo**
- Ad ogni processo viene assegnato dal S.O. un identificativo univoco: il **PID**
- Un processo può essere **attivo** o **sospeso**
- La shell è essa stessa un processo. Quando un comando viene eseguito la shell si sospende in attesa del termine del comando.

- È possibile fare in modo che la shell torni immediatamente attiva eseguendo il comando in **sottofondo** (*background*).
- Un programma può essere eseguito in sottofondo usando il carattere &

```
$ bc &
```

- Nel caso di default si dice che il comando è eseguito in foreground.
- Il processo in foreground e può essere sospeso con la combinazione di tasti CTRL+Z dalla shell.

## Comandi per operare sui processi (2)

- Attivare l'esecuzione in background di processi sospesi

```
bg [PID]
```

- Riprendere l'esecuzione in foreground di processi sospesi o in background

```
fg [PID]
```

- Visualizzare i processi sospesi o in background lanciati dalla shell

```
jobs
```

# Comandi per operare sui processi

- Visualizzare informazioni sui processi

```
ps [opzioni...] [PID]
```

- Eliminare un processo

```
kill [opzioni...] PID
```

- Monitorare l'utilizzo delle risorse da parte dei processi

```
top [opzioni]
```



## Comandi rel. allo stato del sistema

- Visualizzare informazioni sul sistema in uso

```
uname [opzioni...]
```

- Visualizzare data e ora

```
date [opzioni...]
```

- Visualizzare informazioni sull'utente

```
id [opzioni...] [nome_utente]
```

- Visualizzare stato di occupazione dei dischi

```
df [opzioni...] [file_system]
```

- Visualizza occupazione di una cartella (-k in Kbytes)

```
du [opzioni] [pathname]
```

uname -a mostra tutte le info

## Posizione dei comandi nel f.s.

- Il comando **type** oppure anche **which**, visualizza il path assoluto di un comando.
- Sintassi:

```
type nome_comando
```

### Esempio

```
$ type bash  
bash is /bin/bash
```

# Laboratorio di Informatica di Base

## Laurea in Informatica Multimediale

Docente: *Andrea Fusiello*  
[profs.sci.univr.it/~fusiello](http://profs.sci.univr.it/~fusiello)

Lucidi a cura di  
Andrea Colombari, Carlo Drioli e Barbara Oliboni

*Lezione 4*

# Elaborazione di testi



*Testo di riferimento:*  
**M. Bertacca, e A. Guidi**  
**“Introduzione a Linux”**  
**McGrawHill**

## Tool di elaborazione testi

---

- In Linux è particolarmente importante disporre di strumenti efficaci per poter leggere, modificare e scrivere file di testo.
- Molte operazioni di **configurazione e manutenzione** del sistema richiedono la modifica di file testuali.
- I programmi di elaborazione di file di testo storici in Linux sono *vi* ed *emacs*. Ne esistono poi molti altri, per esempio *joe*, e *pico*.

## *vi*: caratteristiche

---

- *vi* è un editor storico presente in tutti i s.o. Unix. È uno dei più complicati da usare!
- E' possibile spostare il cursore nel file e fornire comandi tramite combinazioni di tasti.
- Può operare in una delle seguenti modalità per volta: **comando**, **testo**, o **editor di linea**.

## *emacs*: caratteristiche

---

- **emacs** si comporta in modo molto più simile ad un moderno programma di videoscrittura
- In emacs non esistono modalità distinte di funzionamento come in vi.
- I comandi sono invocati tramite combinazioni dei tasti **CTRL**, **ALT** e **ESC** con altri caratteri.

# Elaborazione di testi con *vi*

## ■ Modalità operative di *vi*

[ESC]	[ ]	Modalità comando	I caratteri rappresentano comandi per il movimento del cursore, lo scorrimento pagine e il cambio modalità
		Modalità testo	I caratteri sono inseriti nella posizione del cursore
		Modalità editor di linea	Permette di impostare comandi globali e comandi che agiscono sul testo in modo non interattivo

Transitions: [ESC] ↑ [a] (Command to Text), [ESC] ↓ (Text to Command), [ ] → (Text to Line Editor)

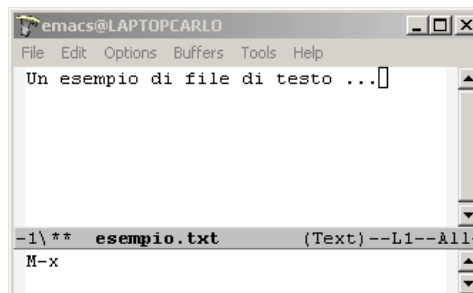


## Elaborazione di testi con *vi* (2)

- Creare un file o aprirne uno esistente
  - `$ vi nomefile`
- Modifiche al testo
  - Inizialmente *vi* si trova in **modalità comando**, è possibile operare modifiche con:
    - **a** (append): va in modalità testo e aggiunge caratteri (ESC per modalità comando)
    - **i** (insert): va in modalità testo e inserisce caratteri (ESC per modalità comando)
    - **x**: cancella il carattere in corrispondenza del cursore (resta in modalità comando)
- Salvare e uscire da *vi*
  - **wq** (se in modalità di linea)
  - **ZZ** (se in modalità comando)

# Elaborazione di testi con *emacs*

- Creare o modificare un documento
  - \$ `emacs nomefile`
- Lo spazio dello screen editor è diviso in tre parti
  - Area di testo
  - Rigia di stato
  - Area di comando



← Area di testo

← Rigia di stato

← Area di comando

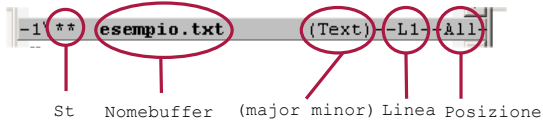
## Elaborazione di testi con *emacs* (2)

- Emacs opera su tre componenti principali:
  - **File:** è un file memorizzato sul disco. Non viene mai manipolato direttamente, tutte le operazioni vengono eseguite copiando i file in dei buffer di memoria e salvando il risultato delle manipolazioni sui buffer in un file.
  - **Buffer:** è una struttura interna che contiene il testo da elaborare. Possono esserci più buffer attivi allo stesso tempo.
  - **Finestre:** una finestra corrisponde alla visualizzazione di un buffer. E' possibile visualizzare uno o più buffer per volta aprendo e chiudendo finestre durante una sessione di elaborazione del testo.

## La riga di stato di *emacs*

- Visualizza informazioni relative al testo corrente.

- Struttura:



- **St**: indica se il file è stato salvato dopo l'ultima modifica.  
“\*\*” (non salvato), “--” (salvato), “%%” (file di sola lettura)
- **Nomebuffer**: indica il nome del buffer corrente
- **(major minor)**: modalità di editing del file.  
**major** fa riferimento a configurazioni di editing per linguaggi particolari (es. Lisp, C, testo semplice, etc.)  
**minor** fa riferimento a modalità di inserimento testo particolari
- **Linea**: numero di linea su cui è posizionato il cursore
- **Posizione**: posizione del cursore in relazione all'inizio del file

## Comandi principali di *emacs*

- In Emacs i comandi vengono invocati attraverso la combinazione dei tasti CTRL o ALT con altri tasti.

Ad esempio per **uscire** da Emacs si può usare la sequenza **CTRL-x CTRL-c**

### Comandi di manipolazione dei file

CTRL-x CTRL-f	apre un file esistente
CTRL-x CTRL-s	salva il file corrente
CTRL-x CTRL-w	salva il file con nome

## Comandi principali di *emacs* (2)

### Comandi di manipolazione dei buffer

CTRL-x b	seleziona un buffer attivo o crea un buffer nuovo
CTRL-x CTRL-b	elena i buffer attivi
CTRL-x k	elimina un buffer

### Comandi di manipolazione delle finestre

CTRL-x o	seleziona un'altra finestra tra quelle attive
CTRL-x 0	chiudi la finestra corrente
CTRL-x 1	chiudi tutte le finestre eccetto quella corrente
CTRL-x 2	divide la finestra del buffer corrente in 2 (vert.)
CTRL-x 3	divide la finestra del buffer corrente in 2 (orizz.)
CTRL-v	scorrimento del testo in avanti
ALT-v	scorrimento del testo all'indietro

## Comandi principali di *emacs* (3)

### Comandi di spostamento del cursore

CTRL-a	sposta il cursore a inizio riga
CTRL-b	sposta il cursore a sinistra di 1 carattere
CTRL-n	sposta il cursore alla riga sottostante
ESC 6 CTRL-b	cursore a sinistra di 6 caratteri
ESC <	sposta il cursore a inizio buffer
ESC >	sposta il cursore a fine buffer

## Comandi principali di *emacs* (4)

### Comandi di selezione di blocchi

CTRL-barra spazio	segna l'inizio del blocco
ESC h	definisce come blocco il paragrafo corrente
CTRL-x CTRL-p	definisce come blocco la pagina
CTRL-w	cancella un blocco
ESC w	copia un blocco in un buffer di memoria



## Comandi principali di *emacs* (5)

### Comandi di cancellazione

CTRL-d	cancella il carattere a destra del cursore
BACKSPACE	cancella il carattere a sinistra del cursore

### Comandi di cancellazione con memorizzazione

CTRL-k	cancella la parte della riga a destra del cursore
ESC d	cancella parola dopo il cursore
ESC BACKSPACE	cancella parola prima del cursore
CTRL-y	inserisce dopo il cursore il testo cancellato
CTRL-x u	annulla il comando precedente

# Esempi

- **Esempio:** Sequenza che sposta la riga corrente in alto di tre righe.

```
CTRL-k  
ESC 3 CTRL-p  
CTRL y
```

- **Esempio:** Operazioni su paragrafi.

```
ESC h CTRL-w      cancella un paragrafo  
ESC h ESC w      copia un paragrafo
```

- **Esempio:** Copia e incolla. Sequenza che copia la pagina corrente nel buffer di memoria e la incolla all'inizio del file.

```
CTRL-x CTRL-p ESC w  
ESC <  
CTRL-y
```

## Comandi principali di *emacs* (6)

### Comandi di ricerca di stringhe

CTRL-s	cerca un stringa in avanti
ESC CTRL-s	cerca un'espressione regolare in avanti
ESC x replace	esegue una sostituzione globale
ESC x replace-regexp regolari	esegue una sostituzione con espressioni regolari
ESC %	esegue una sostituzione condizionale (query-replace)

# Esempio

```
ESC % stringa_1 [Invio] stringa_2 [Invio] opzione
```

- Questa sequenza di comandi permette di sostituire le occorrenze nel testo di **stringa\_1** con **stringa\_2**, con una procedura interattiva. Dopo il secondo comando di [Invio], all'utente viene chiesto di selezionare per l'occorrenza corrente un'opzione fra le seguenti:
  - **Y o barra spazio**  
Sostituisce e passa alla prossima occorrenza
  - **N o Canc**  
Non sostituisce e passa alla prossima
  - **^**  
Salta all'occorrenza precedente
  - **.**  
Sostituisce l'occorrenza ed esce