# Laboratorio di Metodi Informazionali

Laurea in Bioinformatica

Docente: *Carlo Drioli*

Web: www.scienze.univr.it/fol/main?ent=oi&id=39988

Lucidi a cura di

Andrea Colombari, Carlo Drioli e Barbara Oliboni

(colombari@sci.univr.it drioli@sci.univr.it oliboni@sci.univr.it)

*Lezione 2*

# GNU: un po' di storia…

- **GNU** è un acronimo "ricorsivo" e significa **GNU's Not UNIX**. Proprio perché nasce con l'idea di sviluppare un S.O. stile UNIX ma libero, che permettesse di sviluppare liberamente favorendo così la collaborazione tra programmatori.
- Il Progetto GNU (1983, **Richard Stallman**), si basa sulla gestione dei diritti d'autore secondo la definizione di **software libero** (contrapposta a software proprietario).
- Fulcro del Progetto GNU è la licenza GNU General Public License (**GNU GPL**), che sancisce e protegge le libertà fondamentali che, secondo Stallman, permettono l'uso e lo sviluppo collettivo e naturale del software.
- Nel 1984 inizia lo sviluppo del Sistema GNU. In realtà il kernel di tale sistema Hurd è tuttora in lavorazione. Ma nel 1991 **Linus Torvalds** scrisse il kernel Linux e lo distribuì sotto licenza GNU GPL. I sistemi GNU con kernel Linux vengono ufficialmente chiamati **GNU/Linux**.
- Per poter gestire alcuni casi il Progetto GNU ha creato anche la GNU Lesser General Public License (**GNU LGPL**), che permette di integrare software libero all'interno di software proprietario.

# 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

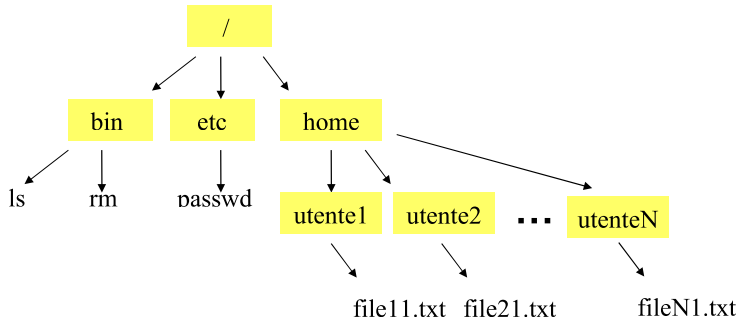
- I dati vengono organizzati in **file**
  - Un file è un contenitore logico di informazioni (dati o istruzioni)
  - Ogni file è identificato da un **identificatore** o **filename** (nome.estensione), dalla **periferica** (drive) e dal **percorso** (path) sulla periferica, da varie altre informazioni (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**

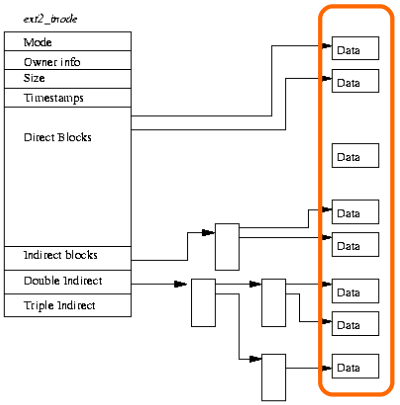
  Esempio: cammino assoluto e cammino relativo rispetto alla directory utente1

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

$ rm subdir1/file1.txt
```

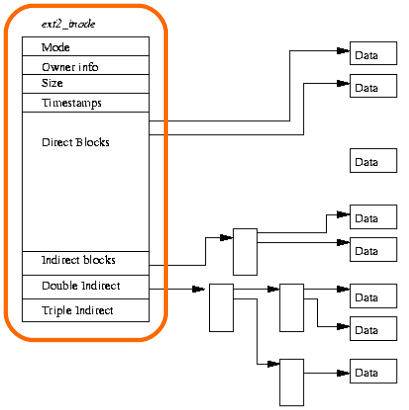
# Organizzazione fisica

- 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)



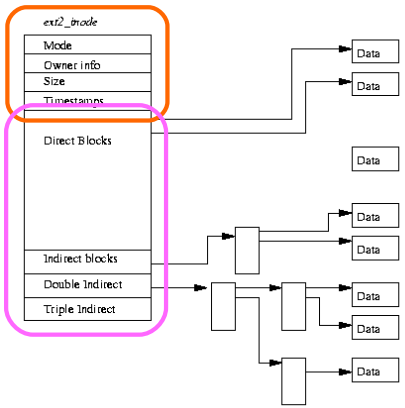

# Organizzazione fisica

- 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**)



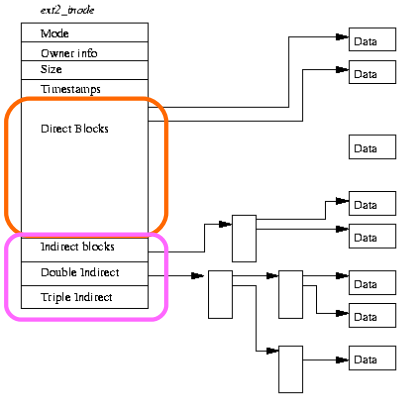

# Organizzazione fisica

- 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**

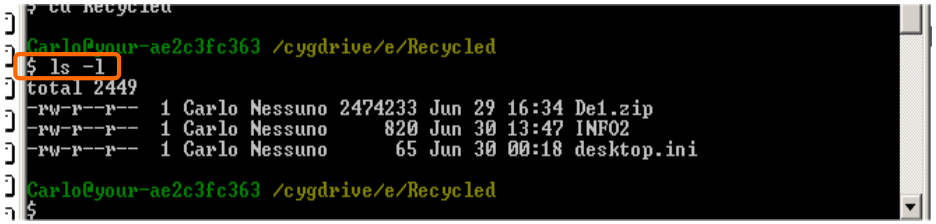


# Permessi e protezioni

- 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

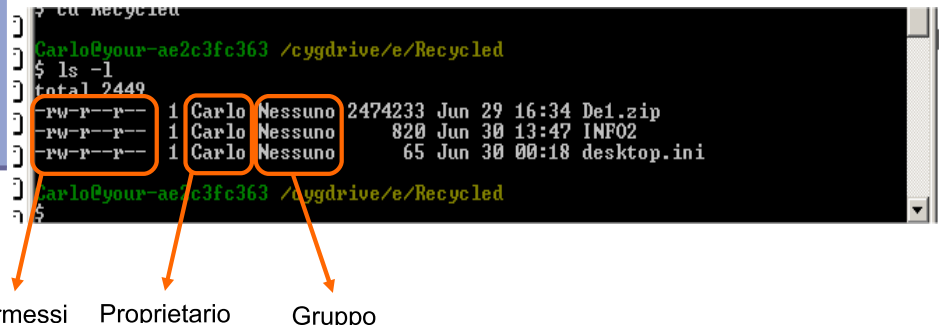
# Permessi e protezioni (2)

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




# Permessi e protezioni (2)

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




# Codifica dei permessi

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


- `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 pipe: premessa
## *Redirezione*

- Un programma Linux per gestire i dati di un file deve richiedere al sistema di aprire un flusso di comunicazione tra il programma e il file.
  - Flusso di input

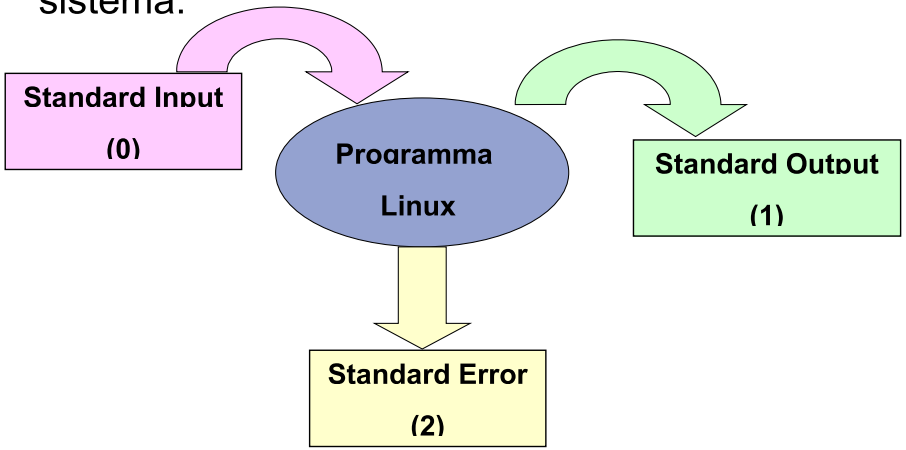
    Il programma può solo leggere il contenuto del file.
  - Flusso di output

    Il programma può solo scrivere nel file.
  - Flusso di input/output

    Il programma può sia leggere che scrivere nel file.

# File di tipo pipe: premessa
## *Redirezione (2)*

- Un programma Linux quando viene eseguito ha sempre tre flussi preventivamente aperti dal sistema.




# File di tipo pipe: premessa
## *Redirezione (3)*

- I tre flussi sempre aperti di un programma si possono *redirigere* da o verso un file.
  - Un programma può leggere le informazioni di cui necessita da un file piuttosto che da tastiera (standard input).

```
$ ls -l >lista
$ ls -l >>lista
$ echo 3 + 4 >conto
```

  - Un programma può scrivere le informazioni che produce su un file piuttosto che sul video (standard output).

```
$ bc < conto
7
```

  - Un programma può scrivere i messaggi di errore su un file piuttosto che sul video (standard error).

```
$ ls -l /akxa 2>lista
```

# File di tipo pipe: premessa
## *Convogliamento (piping)*

- Linux possiede un meccanismo di comunicazione tra due processi che permette di convogliare direttamente l'uscita di un file ottenuta da un processo sull'ingresso di un file in un altro processo.
- Risorsa che permette questo tipo di comunicazione: pipe (condotto).

```
$ ls -l /bin |more
$ ls -l /bin |grep a|more
```

- La pipe permette solo uno scambio di dati unidirezionale.
- I processi possono avere più pipe aperte contemporaneamente.
  - ➔ Si possono realizzare comunicazioni bidirezionali.

# 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**

| | |
|---|---|
| `$ mkfifo fifo1 fifo2` | Crea due file FIFO: fifo1 e fifo2 |
| `$ cat mio file >> fifo1 &` | Invia mio file a fifo1 senza attendere (&) |
| `$ sort < fifo1 >> fifo2 &` | Esegue il riordino di quanto ottenuto da fifo1 e invia il risultato a fifo2 senza attendere (&) |
| `$ lpr < fifo2` | Accoda la stampa di quanto ottenuto da fifo2 |

# 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

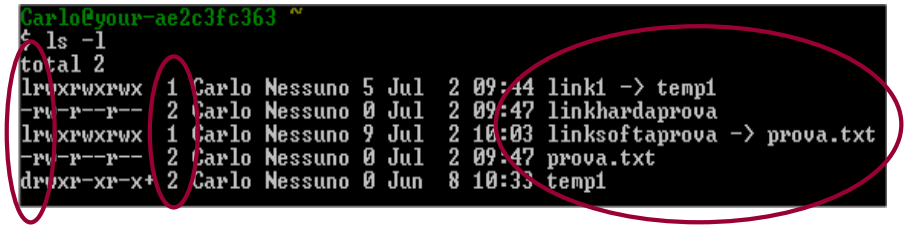
```
ln [-opzioni] nomefile nomelink
```

- Sintassi del comando per un soft link

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

# Visualizzazione dei link

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




# Eliminazione di link

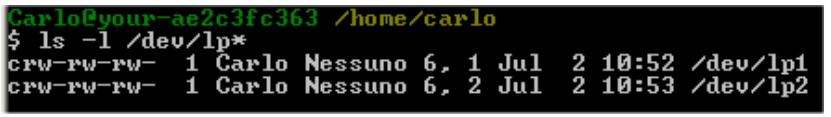
- Con il comando `$ rm nomelink` è possibile cancellare un link
- Nel caso di **hard link**: il comando provoca un decremento del numero di riferimenti nell' i-node del file collegato. Quando questo numero assume valore zero, il file è rimosso dal disco e l' i-node viene reso disponibile per altro utilizzo
- 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, maior number (classe/tipo del device) e minor number (identifica un device all'interno di una classe), che identificano la periferica:



# 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*).
- Si noti che un file prodotto con un elaboratore di testi **"evoluto"**, cioè con formattazione, non è un semplice file di testo, in quanto contiene svariate informazioni in più (tipi dei caratteri, dimensione dei caratteri, ecc.).
- Spesso identificati dall'estensione ".txt" ma non è un obbligatorio. 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 file corti o parte finale

```
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 …]
```

- Ricerca di parole, frasi o espressioni regolari in uno o più file

```
grep [opzioni…] expr [file …]
```

# Operazioni su file di testo: esempi

Si considera il file "miofile" contenente un elenco di parole:

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

Le seguenti "wildcards" possono essere usate nelle *espressioni regolari* per la ricerca di parole nel file con *grep -E*

- * (+) : il carattere precedente compare zero (una) o più volte nel pattern

Esempio:

```
$ grep -E "ca*r" miofile
cr
car
caar
```

# Operazioni su file di testo: grep

- ? : il carattere precedente compare (al più) una volta nel pattern

Esempio:

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

- . : un qualunque carattere

Esempio:

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

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

Esempio:

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

# 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
cr
ca
car
```

# Operazioni su file di testo: grep

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

Esempio:

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

- | : l'espressioni a destra e a sinistra del simbolo vengono entrambe valutate

Esempio:

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

Esempio:

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

# Operazioni su file di testo: grep

- ^ : inizio riga
- $ : fine riga
- \< : inizio parola
- \> : fine parola
- \n: back-reference, rappresenta la sottostringa trovata precedentemente in corrispondenza della n-esima parentesi tonda

Esempio:

```
"([a-z])\1" rappresenta le
doppie
```

- Consideriamo 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**

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

- Righe con parole terminanti in **le**

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

- Righe che contengono esattamente la parola **le**

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

- Righe che iniziano con la parola **le**

```
$ 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 "wildcards" (es. cerco la stringa "pippo.txt"), tale carattere deve essere preceduto da \ (es. cerco "pippo\.txt") altrimenti sarà interpretato come "wildcard" e non come semplice carattere.
- 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 | inzio di una parola |
| \> | B | fine di una parola |
| . | B | un singolo carattere (qualsiasi) |
| [str] | B | un qualunque carattere in `str` |
| [^str] | B | un qualunque carattere non in `str` |
| [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 `exp` 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`).

# Elaborazione di testi

# 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 introdurremo anche ***ioe***.

## *vi*: caratteristiche

- *vi* è un editor di tipo “screen editor”. il terminale viene cioè usato per visualizzare una pagina di testo.
- 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**.

## *ioe*: caratteristiche

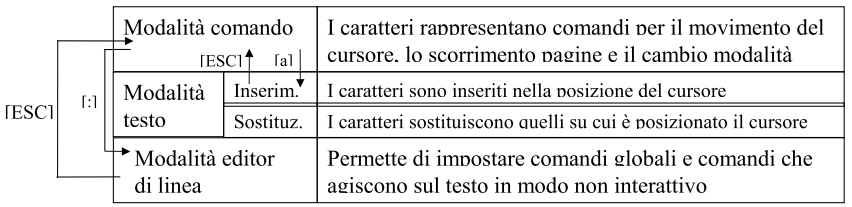
- ***ioe*** è. come *vi.* uno “screen editor”.
- E’ possibile spostare il cursore nel file e fornire comandi tendendo premuto il tasto **CTRL** e combinandolo con altri tasti.

## *emacs*: caratteristiche

- 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***



# 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 *ioe*

- Creare un file o aprirne uno esistente

```
$ ioe nomefile
```

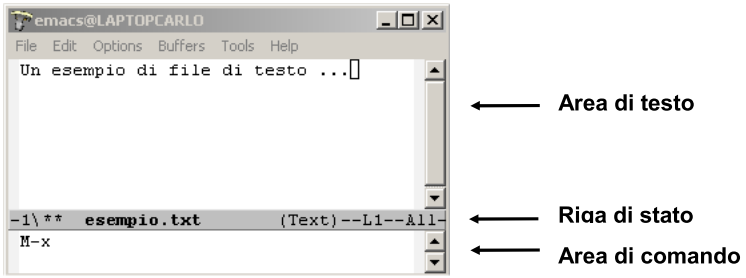
- Alcune combinazioni utili per modificare il testo:
  - **CTRL+KH**: fornisce la schermata di help
  - CTRL+KB: marca l'inizio di un blocco di testo
  - CTRL+KK: marca la fine di un blocco di testo
  - CRTL+KC: copia un blocco di testo marcato
  - CTRL+KM: muove un blocco di testo
  - CTRL+KF: permette di ricercare delle stringhe di caratteri
  - CTRL+KD: permette di salvare
  - CTRL+KX: permette di uscire da ioe salvando le modifiche
  - CTRL+C: esce senza salvare
- Si noti che con ioe si possono utilizzare i tasti freccia, PGUP, PGDN, DEL e BKSP ai quali, volendo, corrispondono combinazioni di tasti opportune (vedi help)

# 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
  - Riga di stato
  - Area di comando

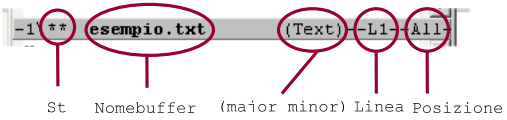



# Elaborazione di testi con *emacs* (2)

- 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 | elenca 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-v | inserisce dopo il cursore il testo cancellato |
| CTRL- | annulla il comando precedente |

# Esempi

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

```
CTRL-k
ESC 3 CTRL-p
CTRL v
```

- 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-v
```

# 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-string | esegue una sostituzione globale |
| ESC x replace-regexp | esegue una sostituzione con espressioni regolari |
| ESC % | esegue una sostituzione condizionale (query-replace) |
| ALT-x occur | cerca un'espressione regolare e salva il risultato |

# Comandi principali di *emacs* (6)

Espressioni Regolari

| | |
|---|---|
| . | Un carattere qualsiasi (almeno uno) |
| * | Ripetizione del carattere precedente zero o piu' volte |
| \{n,m\} | Ripetizione del carattere precedente almeno n e al max. m volte |
| \[…\] | Insiemi di caratteri |
| \[^…\] | Nega l'insieme di caratteri fra parentesi |
| ^ | Inizio della riga |
| $ | Fine della riga |
| \ | Annulla il significato speciale del metacarattere seguente |

# 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