Il File System

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**

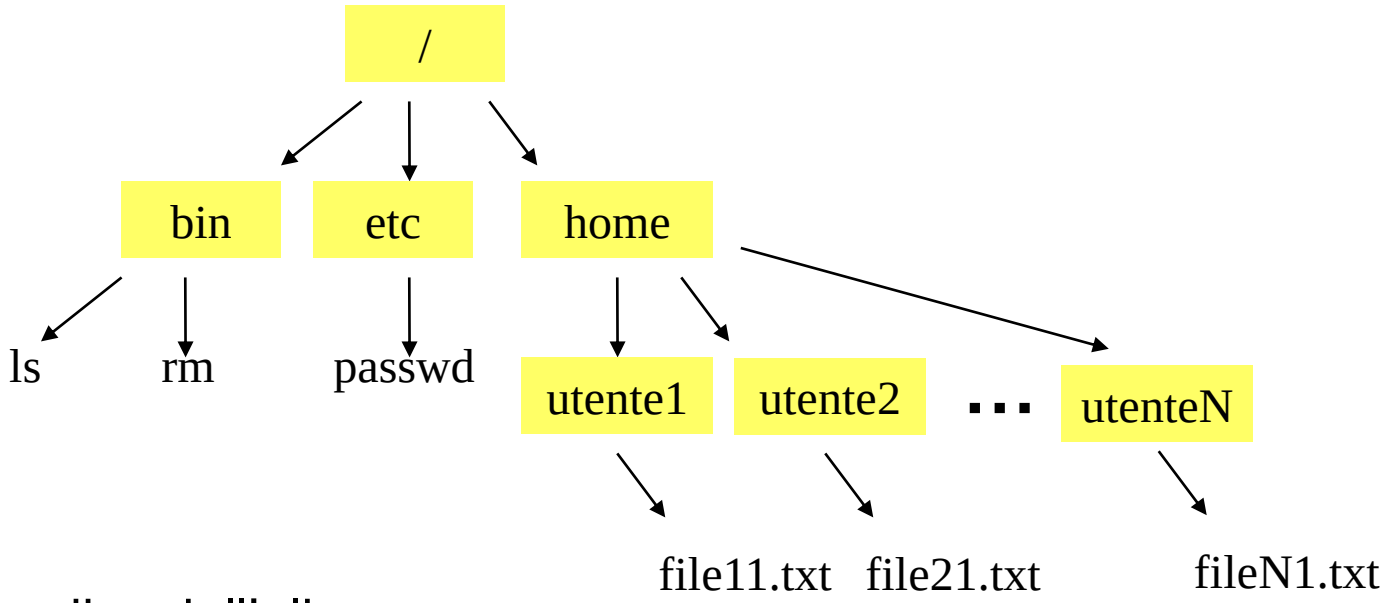
    Dispositivi hardware collegati, vengono visti come file speciali.
  - **pipe**

    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**

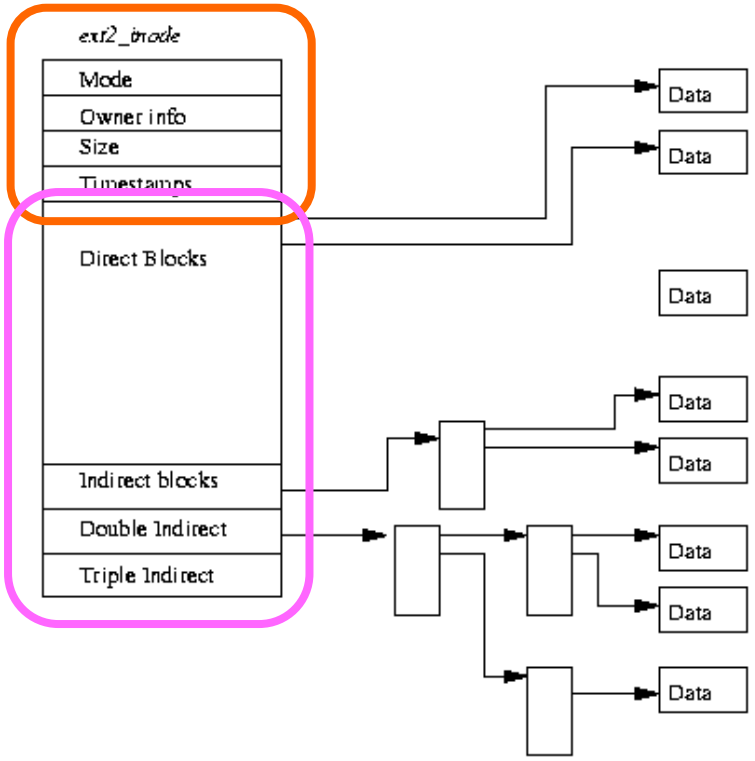
  Esempio: cammino assoluto e cammino relativo rispetto alla directory utente1

```
$ 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)
  - Ogni file è descritto mediante una tabella (**i-node**) che contiene le **informazioni sul file** e i **riferimenti ai blocchi** del file



Comandi principali (1)

- Elencare il contenuto di una cartella

```
ls [opzioni…] [cartella…]
```

  - Opzioni:
    - -l (informazioni estese)
    - -a (visualizza file nascosti, cioè inizianti con il **.**)
    - -R (visualizza sottocartelle)

- Esempio:

```
$ ls -laR
```

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
```

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
```

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 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 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 le linee che contengono una espressione regolare in uno o più file

```
grep [opzioni…] regexp [file …]
```

Operazioni su file di testo

- Contare caratteri, linee e parole

```
wc [opzioni…] [file …]
```

- Oridinamento alfabetico

```
sort [opzioni…] [file …]
```

- Processare testo per colonne

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

Operazioni su file di testo: grep

Si consideri 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
ca
car
caar
```

Operazioni su file di testo: grep

- [$c_1 \ldots c_n$] : un qualunque carattere in $c_1 \ldots c_n$

Esempio:

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

- [c_1-c_n] : un qualunque carattere tra c_1 e c_n

Esempio:

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

- [^$c_1 \ldots c_n$]: un qualunque carattere **non** in $c_1 \ldots c_n$

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
```

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
```

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	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`).

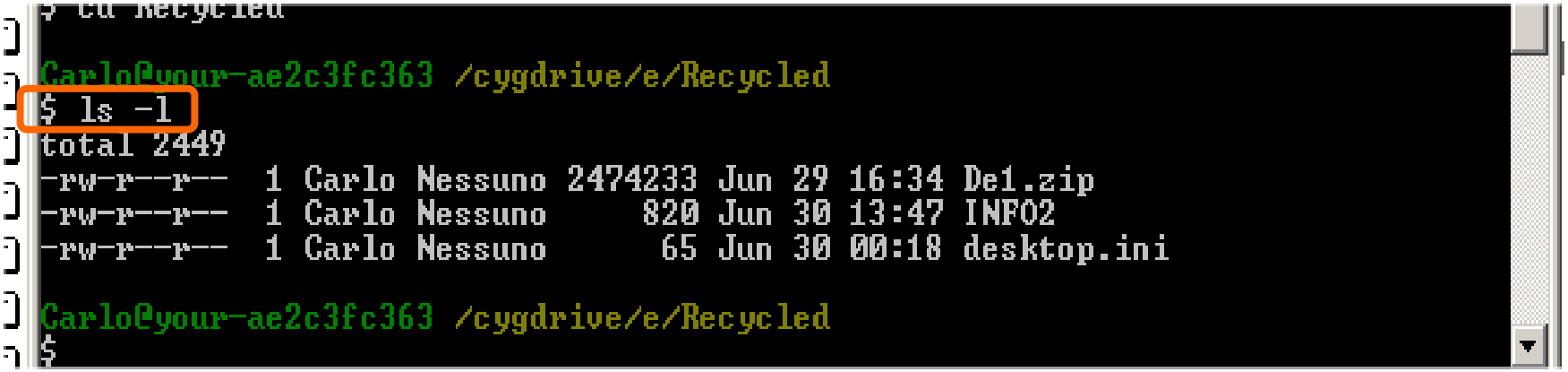
Lavorare sul File System

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

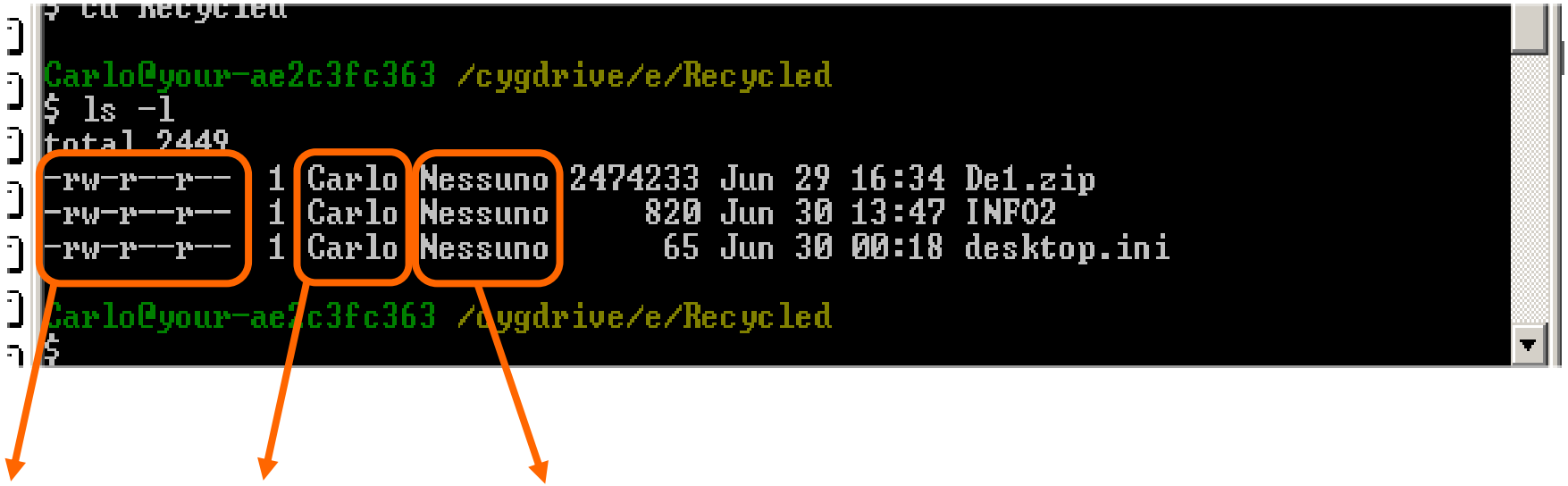
Permessi e protezioni (2)

- 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 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

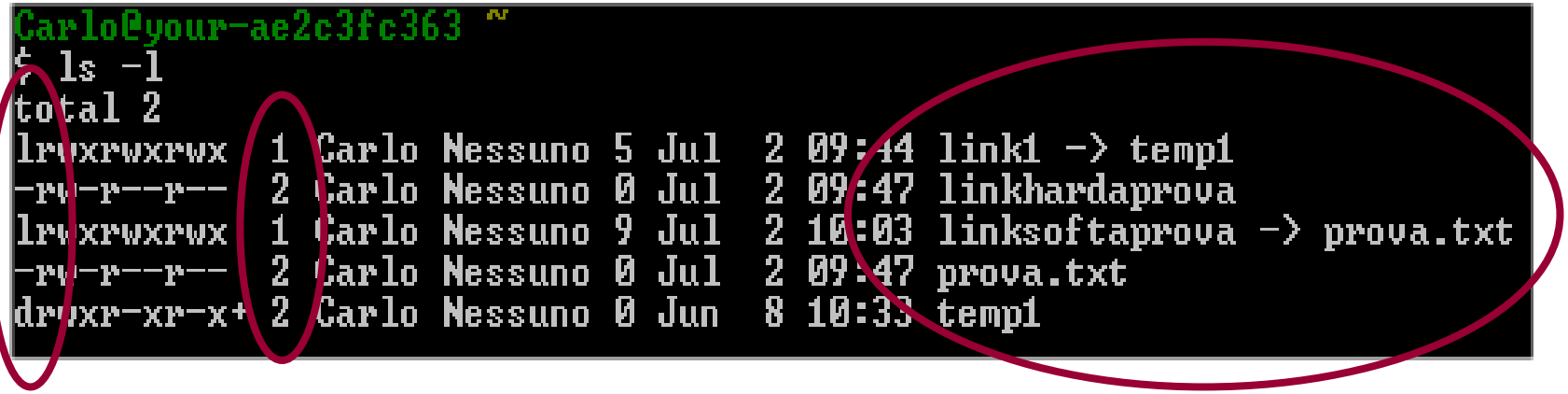
```
ln [-opzioni] nomefile nomelink
```

- 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, 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
```

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 in sottofondo (&)
$ sort < fifo1 >> fifo2 &	Esegue il riordino di quanto ottenuto da fifo1 e invia il risultato a fifo2 in sottofondo (&)
$ lpr < fifo2	Accoda la stampa di quanto ottenuto da fifo2

Stato del sistema

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 attva 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 [JobID]
```

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

```
fg [JobID]
```

- 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]
```

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
```

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 ***joe,*** *e* ***pico***.

vi: caratteristiche

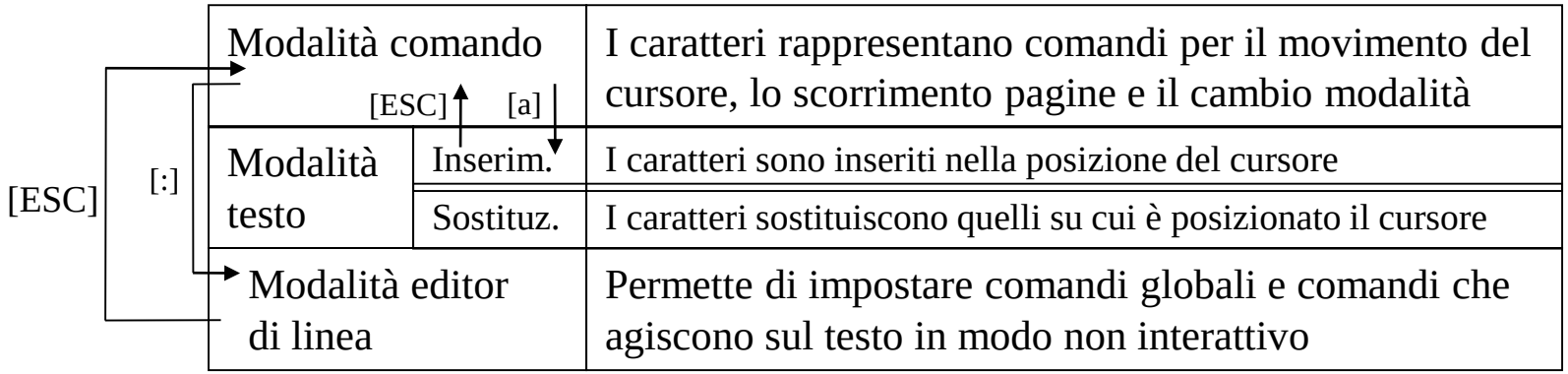
- *vi* è un editor storico presente in tutti i s.o. Unix. È uno dei più complicati da usare!
- E' possibile spostare il cursore nel file e fornire comandi tramite combinazioni di tasti.
- Può operare in una delle seguenti modalità per volta: **comando**, **testo**, o **editor di linea**.

emacs: caratteristiche

- ***emacs*** si comporta in modo molto più simile ad un modermo 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***



Elaborazione di testi con *vi* (2)

- Creare un file o aprirne uno esistente

```
$ vi nomefile
```

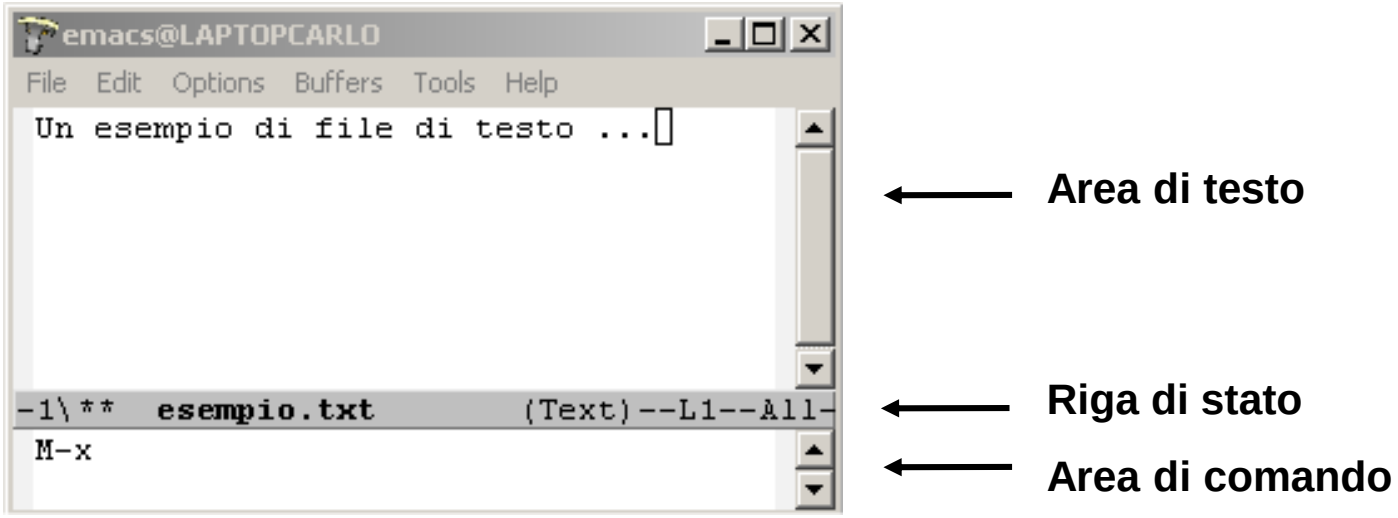
- Modifiche al testo
  - Inizialmente *vi* si trova in modalità comando, è possibile operare modifiche con:
    - **a** (append): va in modalità testo e aggiunge caratteri (ESC per modalità comando)
    - **i** (insert): va in modalità testo e inserisce caratteri (ESC per modalità comando)
    - **x**: 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
  - Riga di stato
  - Area di comando

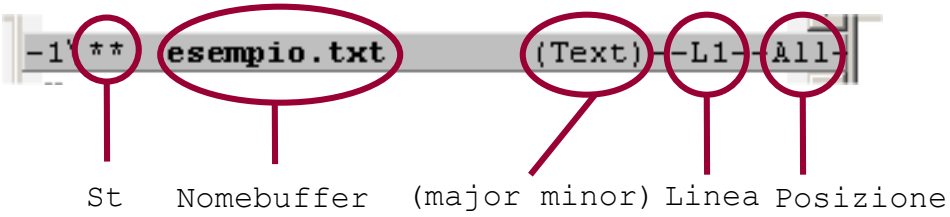


Elaborazione di testi con *emacs* (2)

- 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 successiva
`CTRL-p`	sposta il cursore alla riga precedente
`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`	esegue una sostituzione con espressioni regolari
`ESC %`	esegue una sostituzione con conferma (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

Basic Emacs Reference Card

* EMACS IN 2 PAGES *

SOME NECESSARY NOTATION.
Any ordinary character goes into the buffer (no insert command needed).
In commands, use carriage return, shown as , after a string unless the
chart shows $, meaning to use ESCAPE. ESCAPE is called Altmode in EMACS.

C-	A control character. C-F means "control F". C-_ means "control underscore" (use C-? on a VT100 to get C-_)
M-	A two-character command sequence where the first character is Altmode. M-F means "ESCAPE then F"
M-X string	A command designated "by hand". M-X Revert File means: Altmode then "x" then type "revert file" then
point	EMACS term for cursor position in current buffer

GETTING OUT.

C-X C-S	Write the current buffer into a new version of the current file name
C-X C-W	Write the current buffer into a file with a different name
C-X C-C	Finish by exiting (ask for saving modified buffers)
C-Z	Suspend

BUFFER OPERATIONS.

C-X C-F	Get a file into a buffer for editing
C-X B	Select a different buffer (prompts; default is the last one)
C-X C-B	Display the list of available buffers
C-X K	Kill buffer (prompts for which one; default is current one)
M-<	Move to the top of the current buffer
M->	Move to the end of the current buffer

HELP AND HELPER FUNCTIONS.

C-X U	Undosome previous changes. Repeat this command to undo more changes.
C-G	Abort anything at any time (it beeps, sometimes you need 2)
C-Y	Yank back the last thing killed (kill and delete are different)
M-n	Provide a numeric argument of n for the command that follows
C-Q	Inserts a control character in the buffer (e.g. C-Q C-L)
C-X D	Directory editing subsystem (use ? to see its documentation)

CHARACTER OPERATIONS.

C-B	Move left (Back)
C-F	Move right (Forward)
C-P	Move up (Previous)
C-N	Move down (Next) (usually also arrow keys are available)
BACKSPACE	Delete left
C-D	Delete right
C-T	Transpose previous 2 characters (ht -> th)

WORD OPERATIONS.

M-B	Move left (Back)
M-F	Move right (Forward)
M-BACKSPACE	Kill left (C-Y yanks it back at point)
M-D	Kill right (C-Y yanks it back at point)
M-T	Transpose 2 words around point (if only -> only if)
M-C	Capitalize word
M-U	Uppercase word

LINE OPERATIONS.

C-A	Move to the beginning
C-E	Move to the end
C-O	Open up a line for typing
C-X C-O	Close up any blank lines around point
M-0 C-K	Kill from beginning to point (C-Y yanks it back at point)
C-K	Kill from point to end (C-Y yanks it back at point)

SENTENCE OPERATIONS.

M-A	Move to the beginning
M-E	Move to the end
C-X BACKSPACE	Kill from beginning to point (C-Y yanks it back at point)
M-K	Kill from point to end (C-Y yanks it back at point)

PARAGRAPH OPERATIONS.

M-[	Move to beginning
M-]	Move to end
M-Q	Fill the current paragraph
M-n C-X F	Set the fill column to n (e.g. M-60 C-X F)

SCREEN OPERATIONS.

C-V	Show next screen
M-V	Show previous screen
C-L	Redisplay screen
M-0 C-L	Move the line where point is to line 0 (top) of the screen

PAGE OPERATIONS.

C-X [	Move to beginning (last ^L)
C-X]	Move to end (next ^L)

SEARCH AND REPLACE.

C-S	"Incremental" search searches while you enter string (C-S str$)
C-R	"Incremental" backward search (C-R str$)
M-X Replace String	Replace one string with another
M-X Query Replace	Replace one string with another, wants SPACE meaning "do it" or RUBOUT to skip

REGION OPERATIONS. Region is area of buffer between point and mark (or mark and point).
Some commands set the mark, so check it before using.

C-@	Set the mark (also C-SPACE)
C-X C-X	Interchange point and mark (i.e. go to the other end of the region)
C-W	Kill region (C-Y yanks it back at point)
M-W	Copy region in the kill ring (C-Y yanks it at point)

WINDOW OPERATIONS.

C-X 2	Split the screen in two windows (same buffer shown in each)
C-X 1	Resume single window (using buffer from top window)
C-X O	Move cursor to other window (all the usual commands apply)
M-C-V	Display the next screen in the other window

NOTE: Usually it is possible to select region, kill and yank with mouse.
Function keys (F1...Fn) are usually bound to frequently used operation.
Check your key bindings with command: M-X describe-key.

Basic Emacs Reference Card 10/09/08 15:32 file:///Users/andreafusiello/af/teaching/infobase/emacs2pages.html Pagina 2 di 2