Laboratorio di Informatica di Base

Progetto Tandem 2007/2008

Docente: *Carlo Drioli*

Lucidi a cura di

Andrea Colombari, (colombari@sci.univr.it)

Carlo Drioli drioli@sci.univr.it

e

Barbara Oliboni oliboni@sci.univr.it

Lezione 1

Introduzione al S.O. Linux

Testo di riferimento:

Vincenzo Manca

“Metodi Informazionali”

Bollati Boringhieri

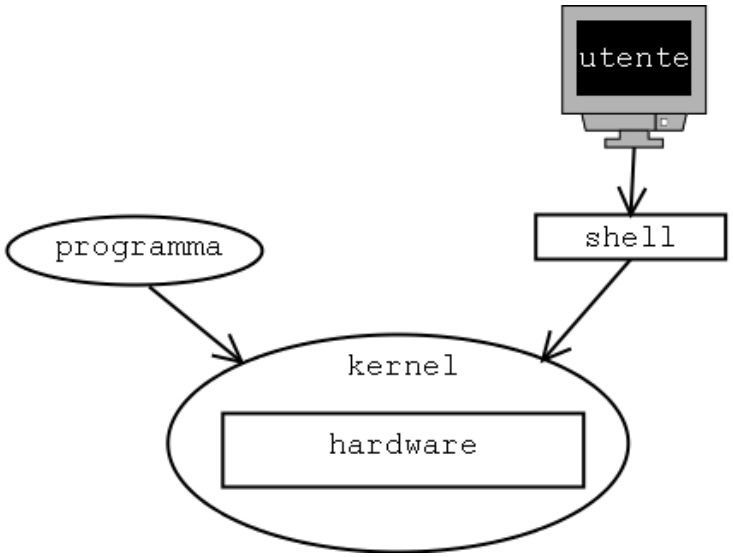
L'elaboratore elettronico

- Un *elaboratore elettronico* deve essere in grado di:
  - eseguire istruzioni su dati e controllare il flusso dell'esecuzione
    → **CPU**, Central Processing Unit:
    - Unità logico aritmetica (**ALU**, Arithmetic Logic Unit)
    - Unità di controllo (**CU**, Control Unit)
    - **Registri**
  - memorizzare le istruzioni e i dati su cui esse operano
    → memoria centrale (**RAM**, Random Access Memory) e **unità disco** (o memoria di massa)
  - interagire con gli utenti e con eventuali altri sistemi
    → dispositivi di ingresso/uscita, i.e. input/output (**I/O**)

Il sistema operativo

- Il sistema operativo fornisce dei servizi ai programmi applicativi e agli utenti rendendo utilizzabili le risorse fisiche presenti nel sistema di calcolo.

- Il sistema operativo può essere inteso come uno strumento che virtualizza le caratteristiche dell'hardware sottostante, offrendo di esso la visione di una macchina astratta più potente e più semplice da utilizzare di quella fisicamente disponibile.



Il S.O. GNU/Linux: introduzione

- GNU/Linux è un sistema operativo libero di tipo Unix, distribuito con licenza GNU GPL (General Public License)
- Il kernel Linux nasce dalla collaborazione a distanza, grazie a Internet, di numerosi sviluppatori volontari
- E' un sistema **multiutente** e **multiprocesso** (**multitasking**)
- Adotta un proprio file system per la memorizzazione dei file (ext2fs) ma e' compatibile con i principali file system in uso (es. MS-DOS, Fat32)
- E' dotato di potenti interfacce a caratteri (**shell** di comando) ma anche di ambienti desktop evoluti come **KDE** e **GNOME**

Il SO Linux: accesso alla macchina

- Per accedere ad una sessione Linux è necessario disporre di un nome utente riconosciuto dal sistema (login) e di una parola d'ordine (password)

```
login: mialogin
password: *******
```

- Dopo l'autenticazione, l'utente accede alla propria cartella di lavoro (home directory, ad es. /home/mialogin) e può interagire con il S.O. con un'interfaccia, ad es. un interprete di comandi (shell)
- Il prompt ($) avvisa l'utente che l'interprete è pronto ad accettare comandi
- Per terminare la sessione di lavoro si possono usare i comandi logout o exit. Il sistema tornerà nello stato di richiesta di login e password

```
$ logout
    oppure
$ exit
```

Il SO Linux: accesso alla macchina (2)

- La shell legge i comandi dell'utente, li interpreta e richiede al sistema l'esecuzione delle operazioni richieste dal comando.

  Esempio: il comando passwd permette di impostare una nuova password:

```
$ passwd
Enter the new password (minimum of 5, maximum of
8 characters)
New Password:
```

Il SO Linux: comandi principali

- In generale la sintassi di un comando Linux è:

```
comando [opzioni] [argomenti]
```

- Un manuale dei comandi descrive l'utilizzo e le caratteristiche di ogni comando. Le pagine del manuale si invocano con man argomento e hanno tutte la seguente struttura comune:

NAME: riporta il nome del comando e una breve descrizione delle sue funzioni
SYNOPSIS: descrive la sintassi del comando
DESCRIPTION: descrive lo scopo e il funzionamento del comando
OPTIONS: riporta il funzionamento di tutte le opzioni
ENVIRONMENT: descrive eventuali variabili d'ambiente che interagiscono con il comando
AUTHOR: note sull'autore del comando
COPYRIGHT: note su copyright
BUGS: eventuali errori o malfunzionamenti noti
SEE ALSO: eventuali altre pagine del manuale a cui fare riferimento

Il File System di un S.O.

Componente fondamentale del S.O., i cui obiettivi sono:

- 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**

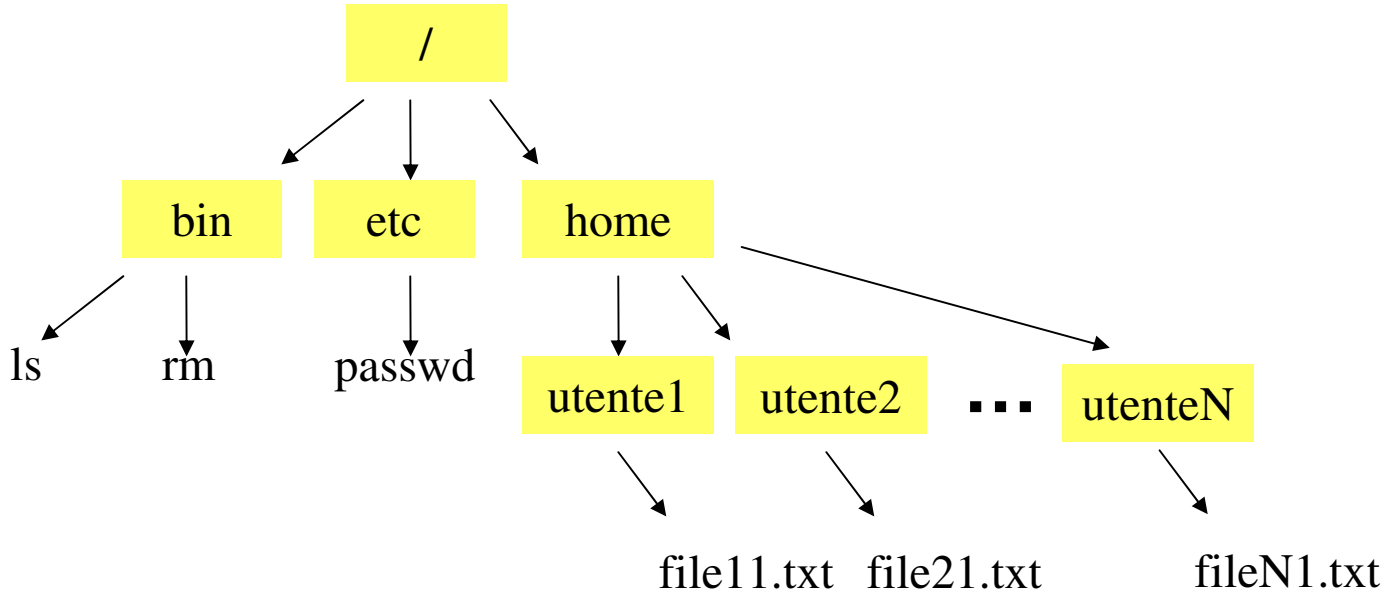
    Dispositivi hardware collegati, vengono visti come file speciali.
  - **pipe**

    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
```

Linux: i comandi principali per operare sul file system

- Elencare il contenuto di una cartella
- Sintassi:

```
ls [opzioni…] [cartella…]
```

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

```
$ ls -laR
```

Linux: i comandi principali per operare sul file system

- Cambiare la cartella corrente
- Sintassi:

```
cd path_nuova_directory
```

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

```
$ cd ..

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

Il SO Linux: comandi principali

- Creare nuove cartelle
- Sintassi:

```
mkdir nome_cartella
```

- Esempio:

```
$ mkdir nuovacartella1 nuovacartella2
```

Il SO Linux: comandi principali (6)

- Copiare file e cartelle

```
cp [opzioni…] sorgente… destinazione
```

- Spostare o rinominare file e cartelle

```
mv [opzioni…] sorgente… destinazione
```

- Visualizzare path assoluto cartella corrente

```
pwd
```

- Eliminare file

```
rm [opzioni…] file
```

- Eliminare una cartella

```
rmdir cartella
```

Caratteri jolly o *metacaratteri*

`*` Sostituisce un insieme di zero o più caratteri qualsiasi

Esempio:

```
$ ls c*
```

`?` Sostituisce un carattere qualsiasi

Esempio:

```
$ ls co??o.txt
```

`[]` Permettono di specificare una lista e/o un intervallo di caratteri possibili

Esempio:

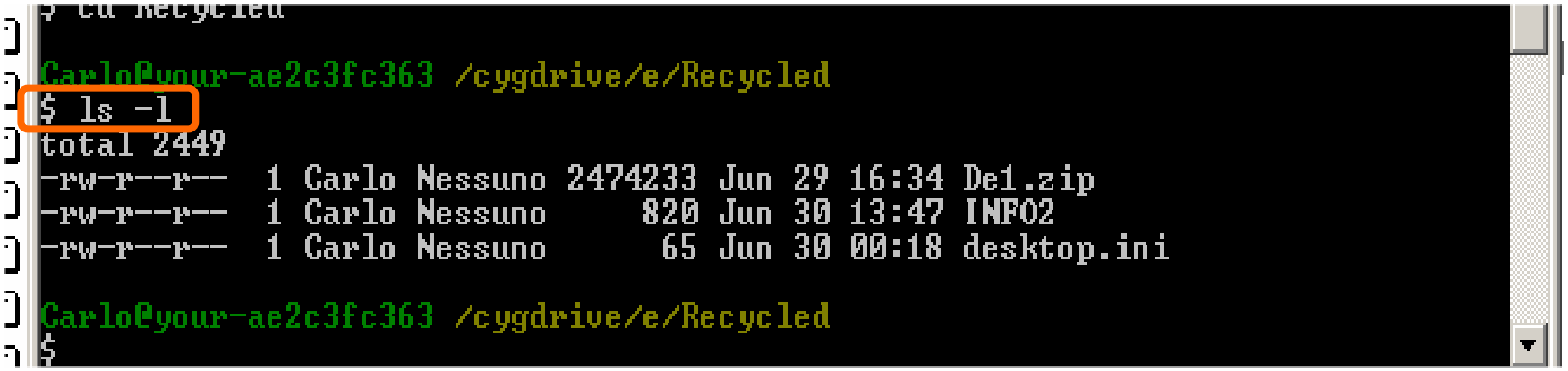
```
$ ls [a-c]*.txt
```

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
```

Flussi di Input e Output: 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

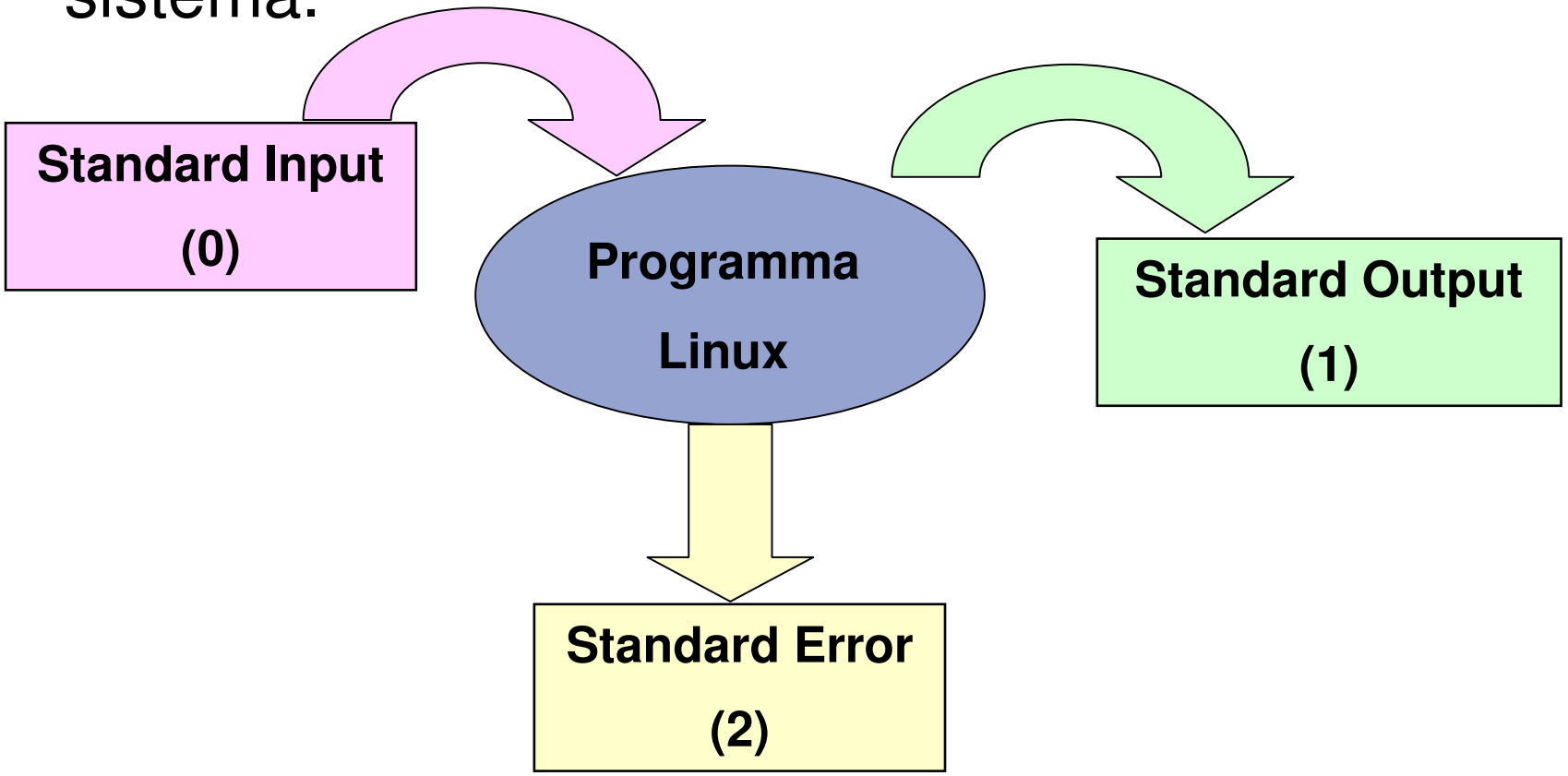
    Il programma può solo leggere il contenuto del file.
  - Flusso di output

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

    Il programma può sia leggere che scrivere nel file.

Flussi di Input e Output: Redirezione

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



Flussi di Input e Output: Redirezione

- 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 /qkxq 2>lista
```

File di tipo pipe: *Convogliamento*

- 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 link

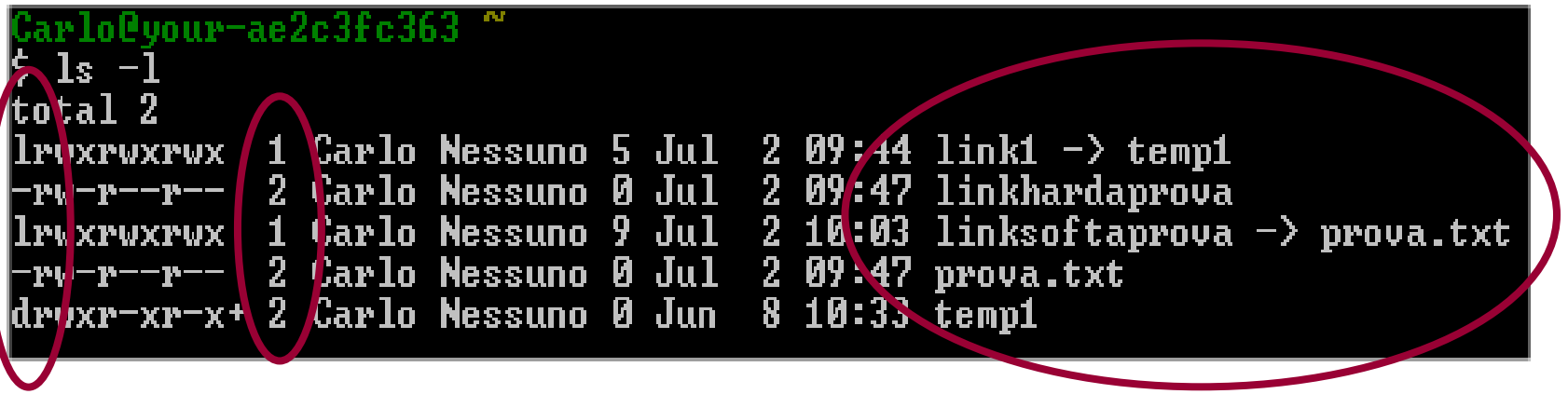
- Lo scopo dei link è potersi riferire a file e directory tramite due o più pathname (link nella home ad un file usato spesso e con path molto lungo)

- 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.

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 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: grep

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
```

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

Esempio:

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

- [] : l'espressione tra parentesi è l'unione delle espressioni contenute

Esempio:

```
$ grep -E'ca[a-z]r' miofile
car
cor
```

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

- . : 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 “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`).

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 un identificativo univoco: PID
- Un processo può essere attivo o sospeso ed eseguito in foreground (fg) o in background (bg)
- Un programma può essere eseguito in bg usando il carattere `&` (`$ ls c* &`) e può essere sospeso con la combinazione di tasti CTRL+Z
- Il sistema operativo fornisce comandi per visualizzare informazioni sui processi e per gestirne l'esecuzione.

Comandi per operare sui processi

- Visualizzare informazioni sui processi

```
ps [opzioni…] [PID]
```

- Eliminare un processo

```
kill [opzioni…] PID
```

- Visualizzare i processi sospesi o in background

```
jobs
```

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

```
fg job_id
```

Comandi per operare sui processi (2)

- Attivare l'esecuzione in background di processi sospesi

```
bg job_id
```

- Monitorare l'utilizzo delle risorse da parte dei processi

```
top [opzioni]
```