Lezione 7
Gestione dei file

Sistemi Operativi (9 CFU), CdL Informatica, A. A. 2022/2023
Dipartimento di Scienze Fisiche, Informatiche e Matematiche
Università di Modena e Reggio Emilia
http://weblab.ing.unimo.it/people/andreolini/didattica/sistemi-operativi

Quote of the day

(Meditate, gente, meditate...)

“I think the major good idea in UNIX was its clean and simple interface: open, close, read and write.”

Ken Thompson (1943-)
Programmatore
Ideatore dei SO UNIX e Plan 9
Ideatore dei linguaggi B e Go
Creatore di Belle (primo motore scacchistico americano)

INTRODUZIONE

Lo scenario

(Uno studente vuole esplorare gli strumenti per la gestione dei file)

Uno studente che sa usare GNOME e la linea di comando vuole cominciare ad esplorare per bene gli strumenti di gestione dei file.

- Uno dei punti di forza di UNIX.
- UNIX venne esteso inizialmente da AT&T per calcolare e stampare i tabulati telefonici degli abbonati.

Interrogativi

(Quali sono le operazioni di base sui file?)

Quali sono i comandi principali con cui l'utente interagisce con un file system?

Creazione? Lettura? Scrittura? Cancellazione? Altro?

FILE E DIRECTORY

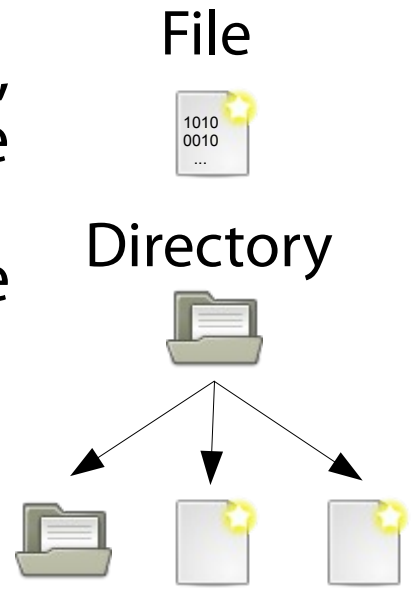
File e directory

(I mattoncini di base)

Nei SO moderni, il paradigma cui si ispirano gli strumenti di memorizzazione e recupero delle informazioni ricorda la scrivania di un ufficio.

Fascicolo (file): contenitore di byte arbitrari, a cui un utente attribuisce un senso tramite le applicazioni.

Cartella (directory): raccoglitore di file e directory.



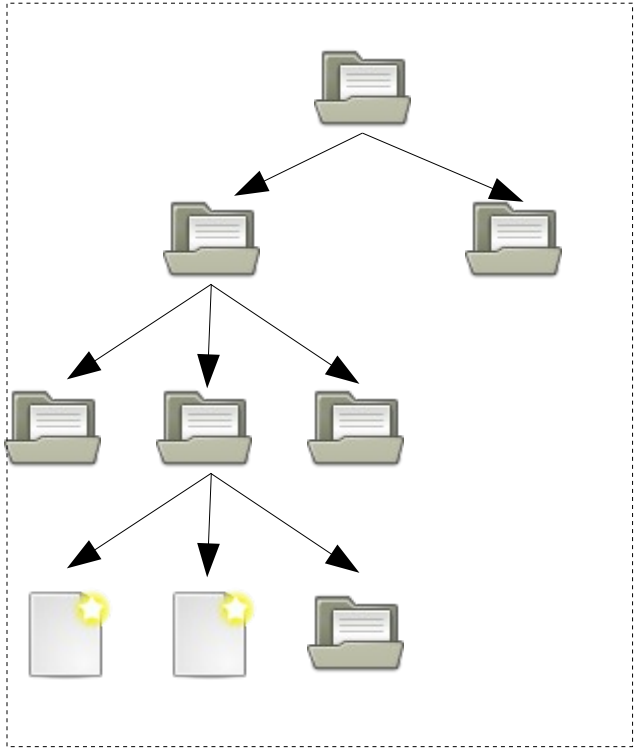
File system

(Organizza una gerarchia di file e directory su un dispositivo di memorizzazione)

Un **file system** è una gerarchia di directory e file, ospitata su un dispositivo di memorizzazione.

Tipicamente, un disco rigido.

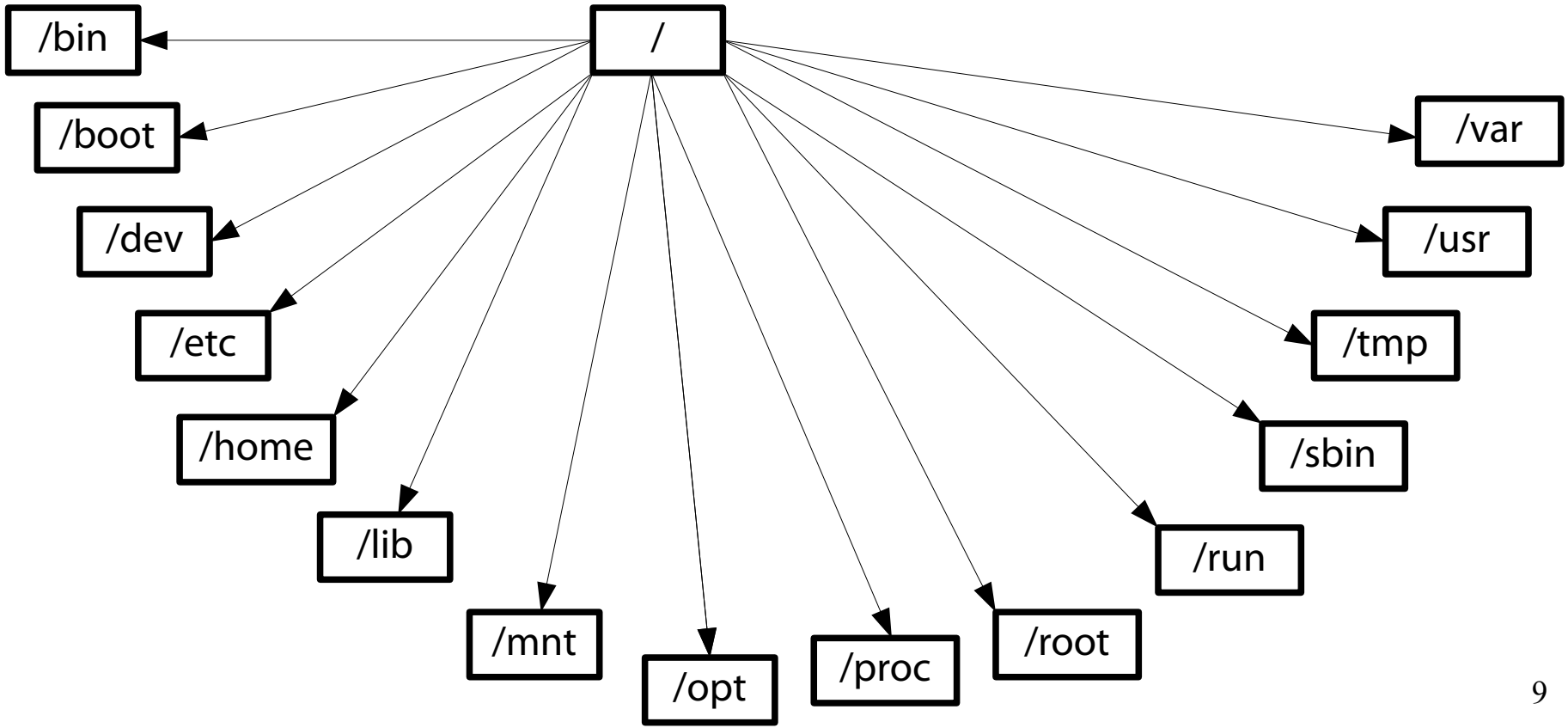
Rappresentazione: **albero** o **grafo orientato**.



File system

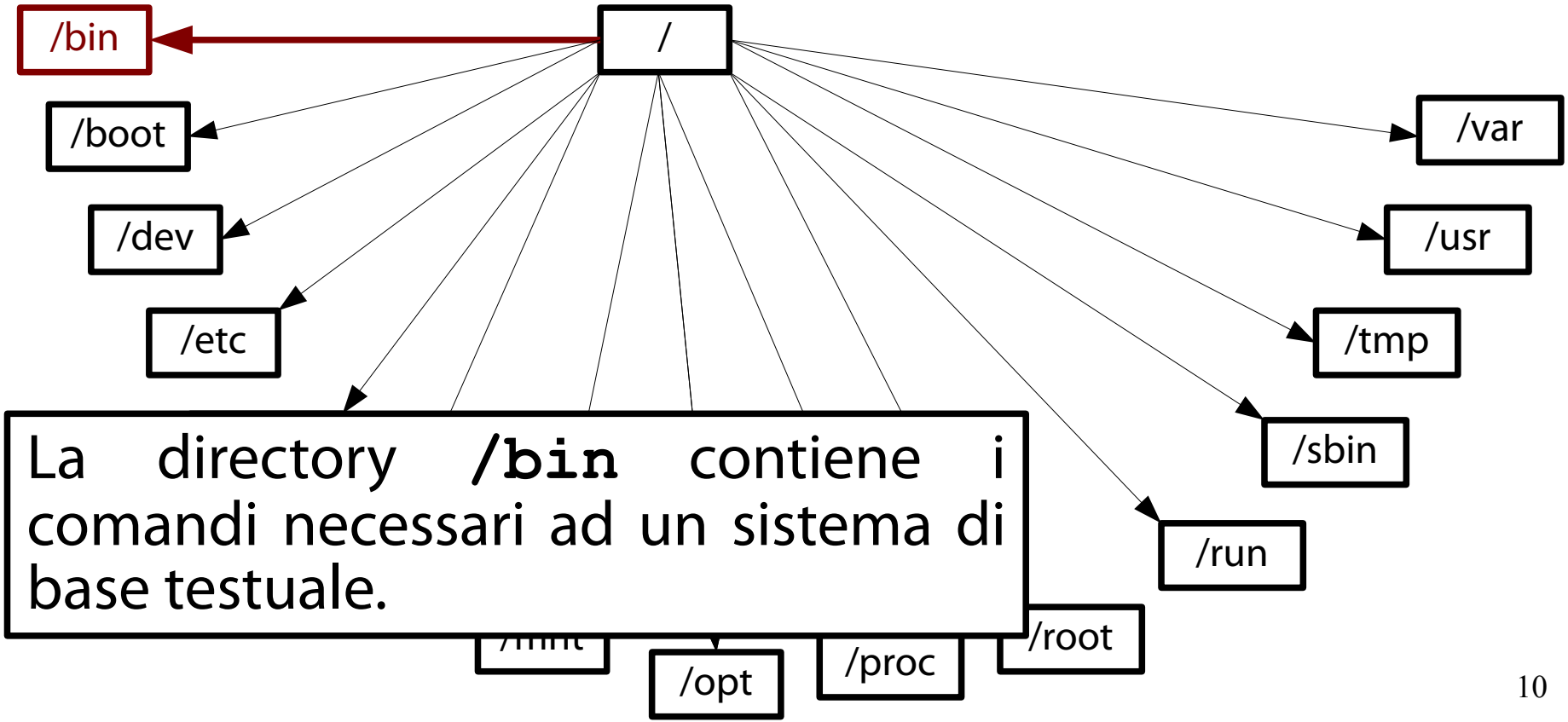
Organizzazione ad alto livello

(10000 feet view)



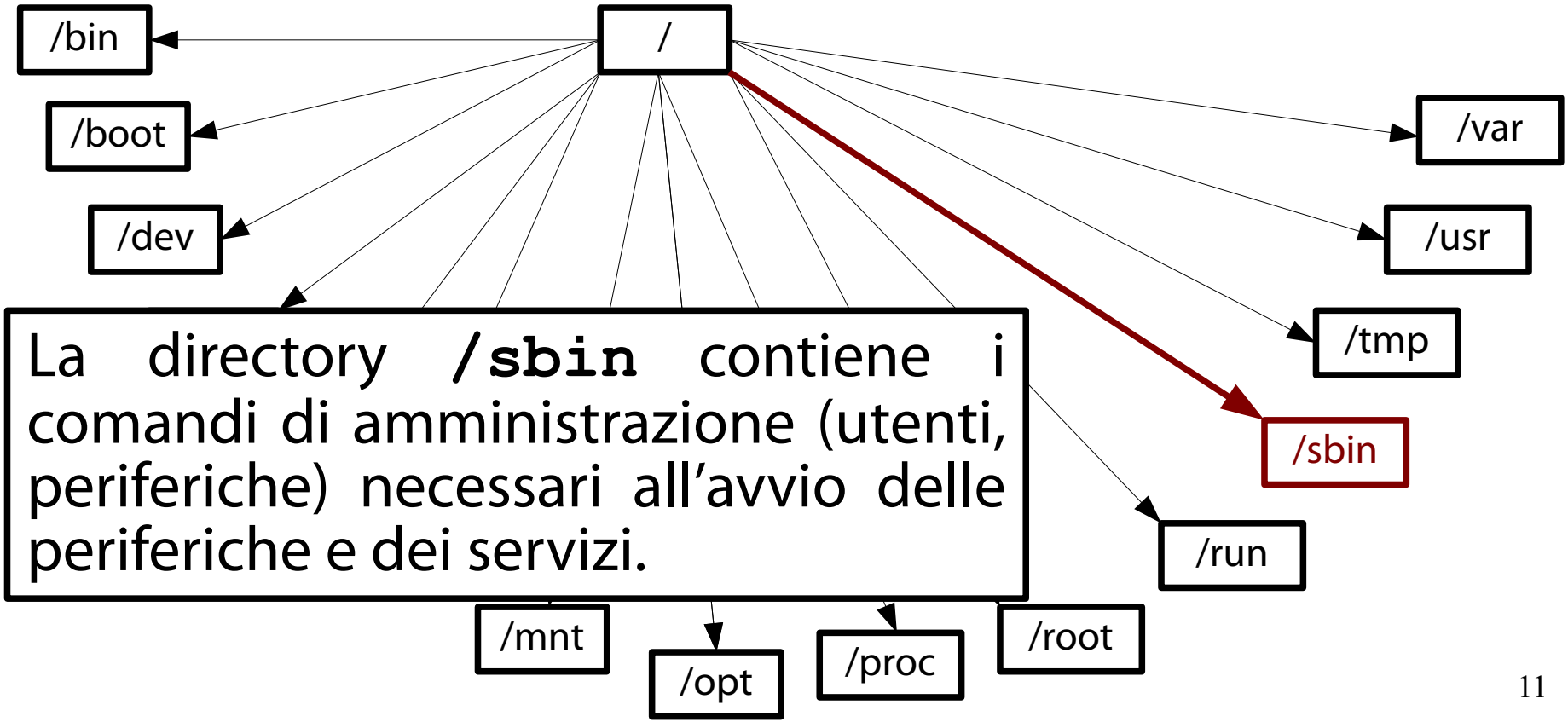
Eseguibili di sistema

(Comandi per l'uso di un SO di base, testuale)



Eseguibili di sistema

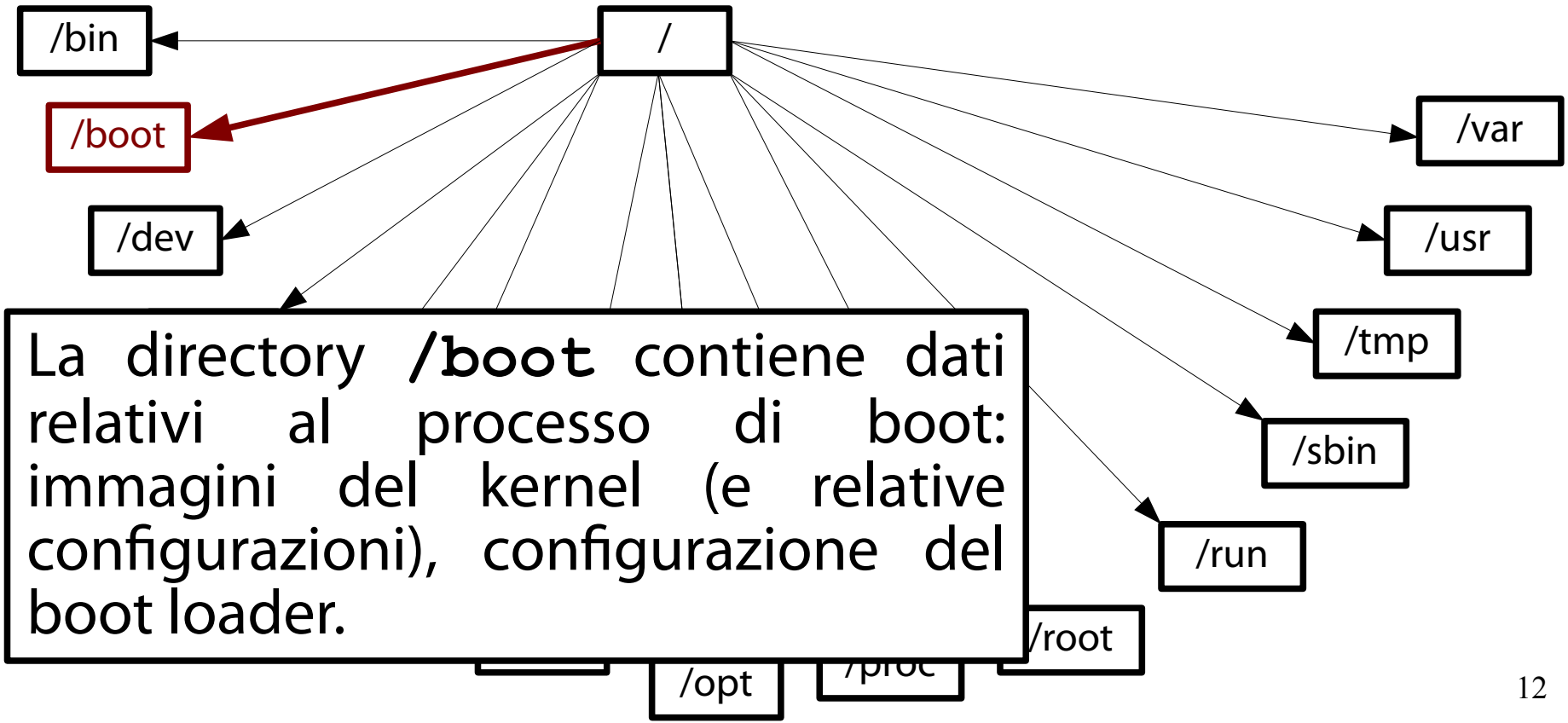
(Comandi per la gestione delle risorse hw e sw



La directory **`/sbin`** contiene i comandi di amministrazione (utenti, periferiche) necessari all'avvio delle periferiche e dei servizi.

Configurazione del boot loader

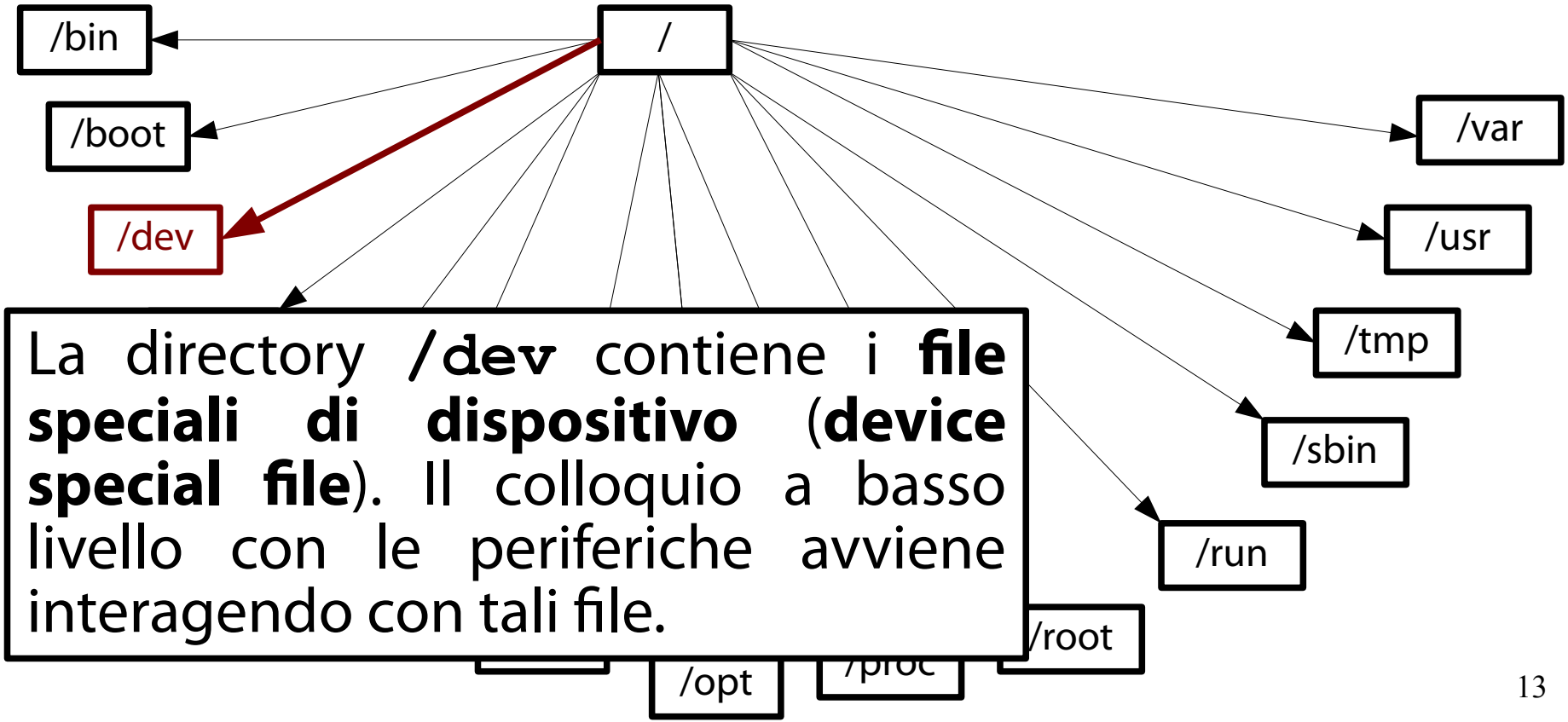
(Immagini del kernel, configurazione del boot loader, RAM disk varie)



La directory **`/boot`** contiene dati relativi al processo di boot: immagini del kernel (e relative configurazioni), configurazione del boot loader.

File speciali di dispositivo

(Consentono l'interazione a basso livello con le periferiche)



La directory **`/dev`** contiene i **file speciali di dispositivo** (**device special file**). Il colloquio a basso livello con le periferiche avviene interagendo con tali file.

File speciali di dispositivo

(Rappresentano periferiche; permettono di colloquiare con esse)

Un **file speciale di dispositivo** è associato ad una periferica e permette di accedervi a basso livello.
I file speciali sono contenuti nella directory **`/dev`**.

```
ls -l /dev
```

I file speciali di dispositivo sono di due categorie.
Caratteri: periferiche lente, accedute serialmente e pochi byte/una riga per volta.
Blocchi: periferiche veloci, accedute serialmente e casualmente, per blocchi di una dimensione minima.

Major number, minor number

(Major → Classe di periferiche; Minor → Specifica istanza di periferica)

Ad ogni file speciale di dispositivo è associata una coppia di numeri interi.

Major number: caratterizza una intera classe di dispositivi (terminale, disco rigido, CD-ROM).

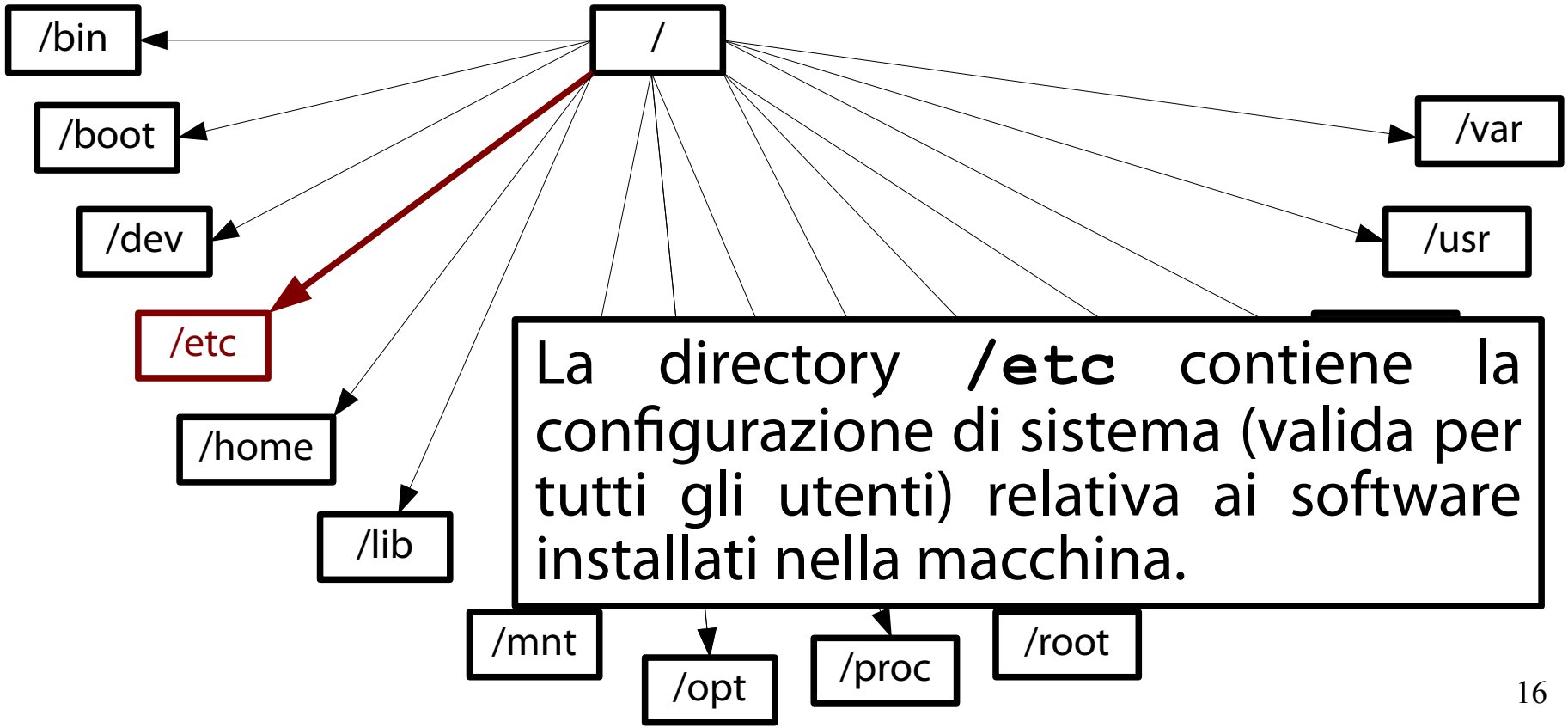
Minor number: caratterizza una specifica istanza di un dispositivo in una classe (il primo disco, il secondo terminale).

Major e minor number sono visibili con il comando:

```
ls -l /dev
```

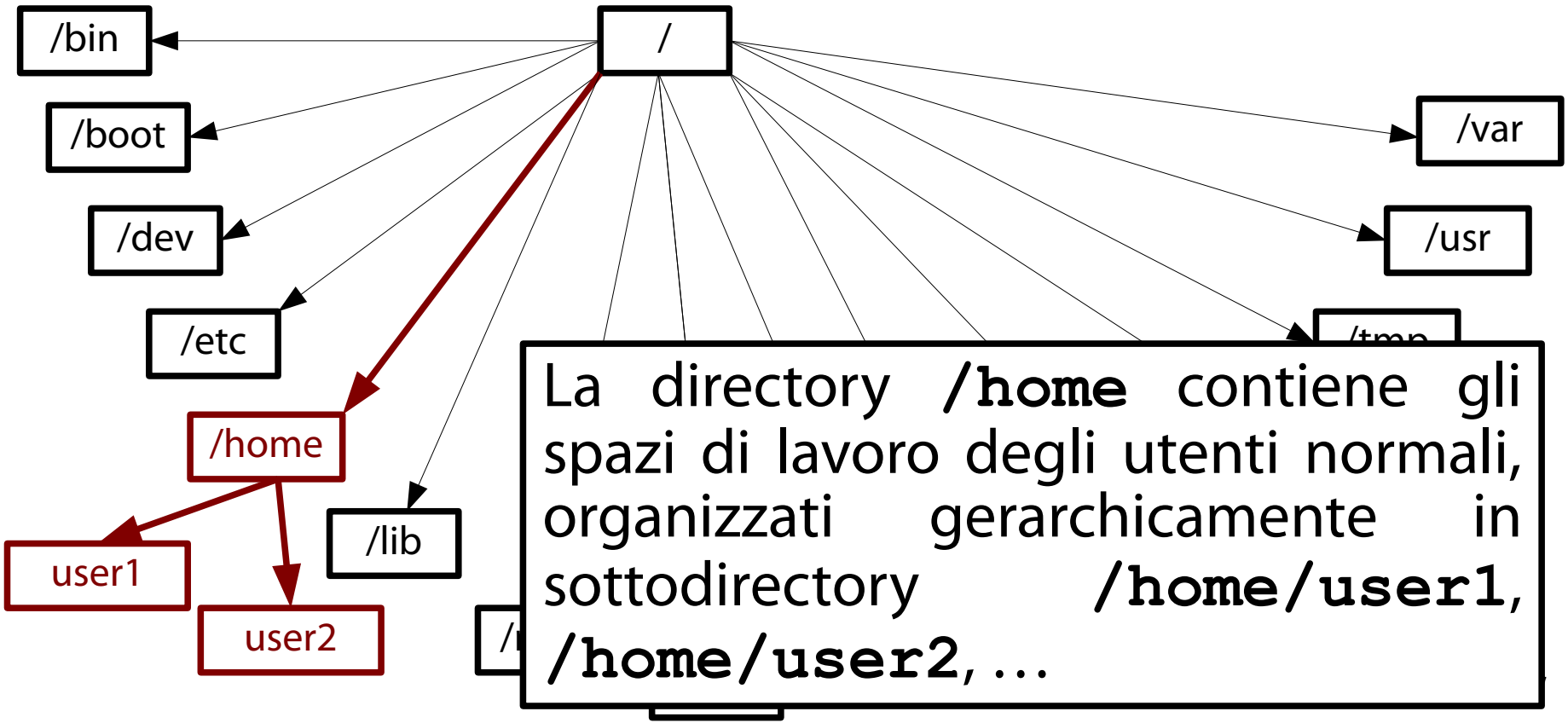
Configurazione delle applicazioni

(File testuali; un file o una directory per applicazione)



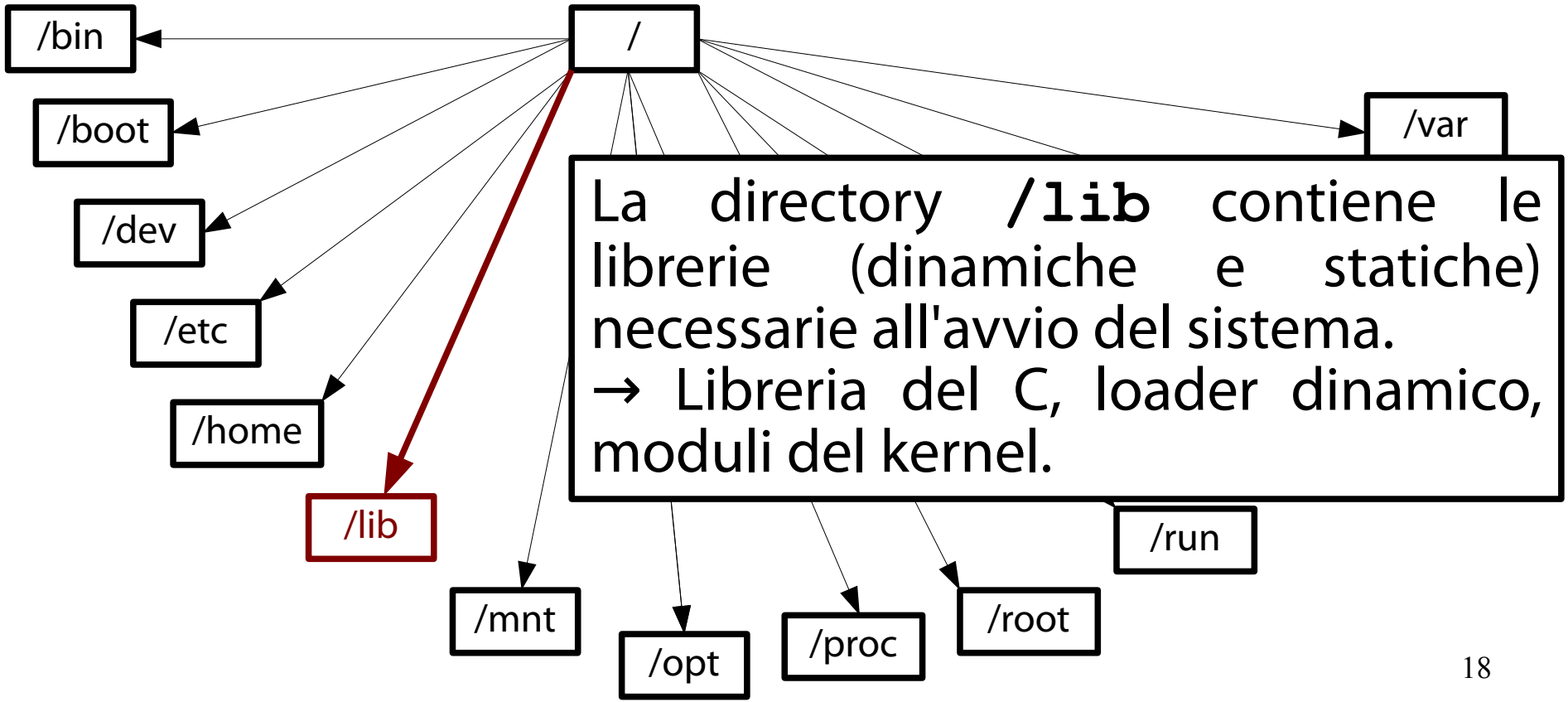
Home utente

(Contengono i file dei singoli utenti)



Librerie di sistema

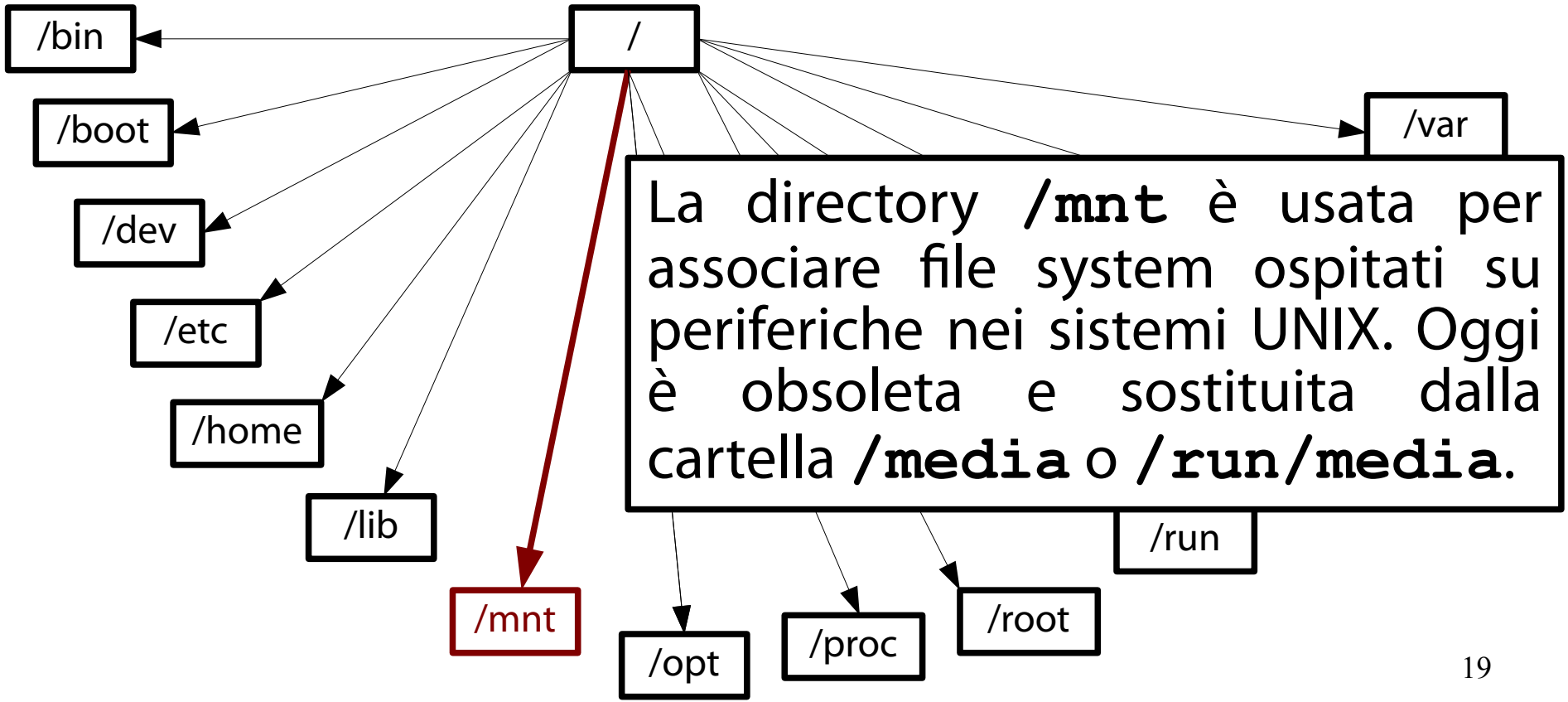
(Usate delle applicazioni di base; condivise tra diversi sottosistemi)



La directory **`/lib`** contiene le librerie (dinamiche e statiche) necessarie all'avvio del sistema.
→ Libreria del C, loader dinamico, moduli del kernel.

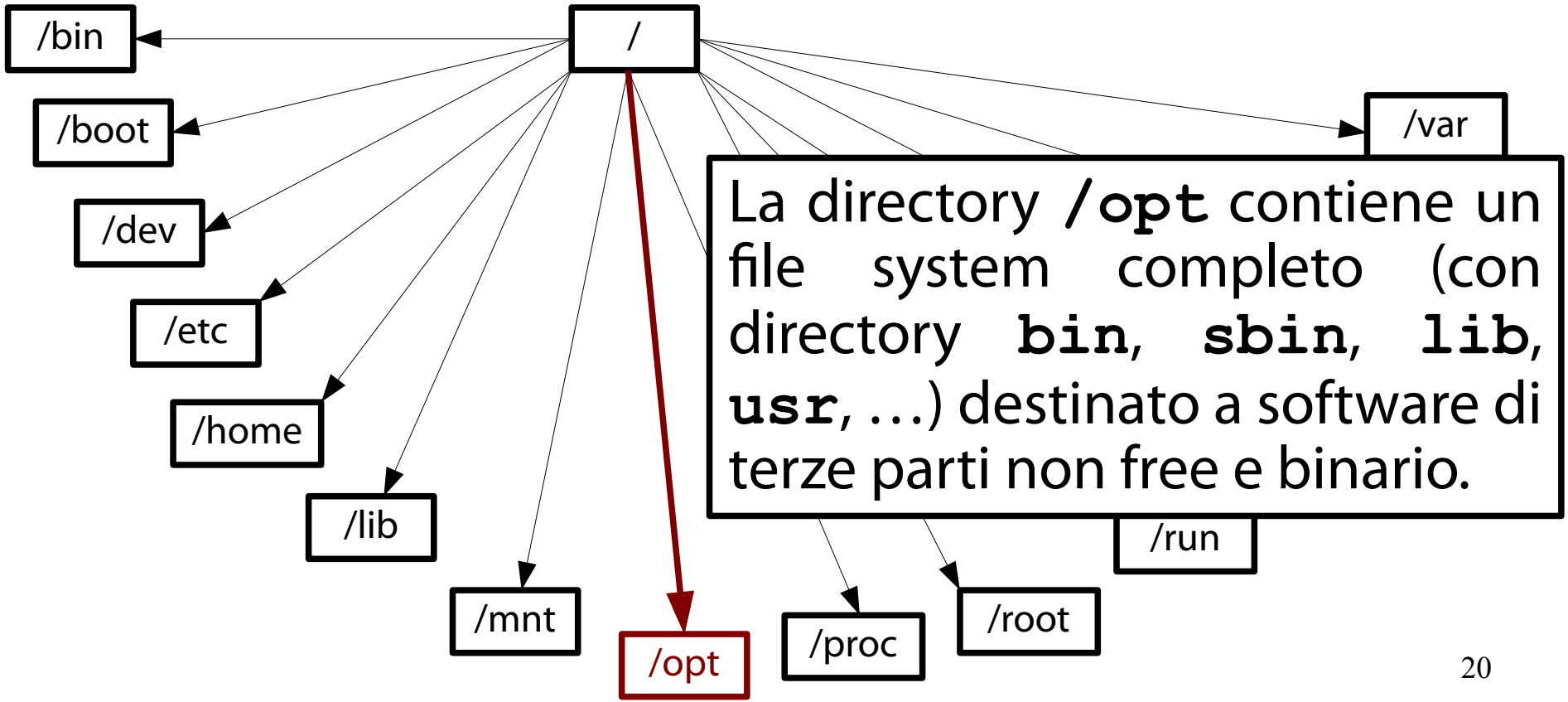
Punto di montaggio dei file system

(I file system ospitati su dispositivi esterni vengono associati a questa directory)



Applicazioni proprietarie third party

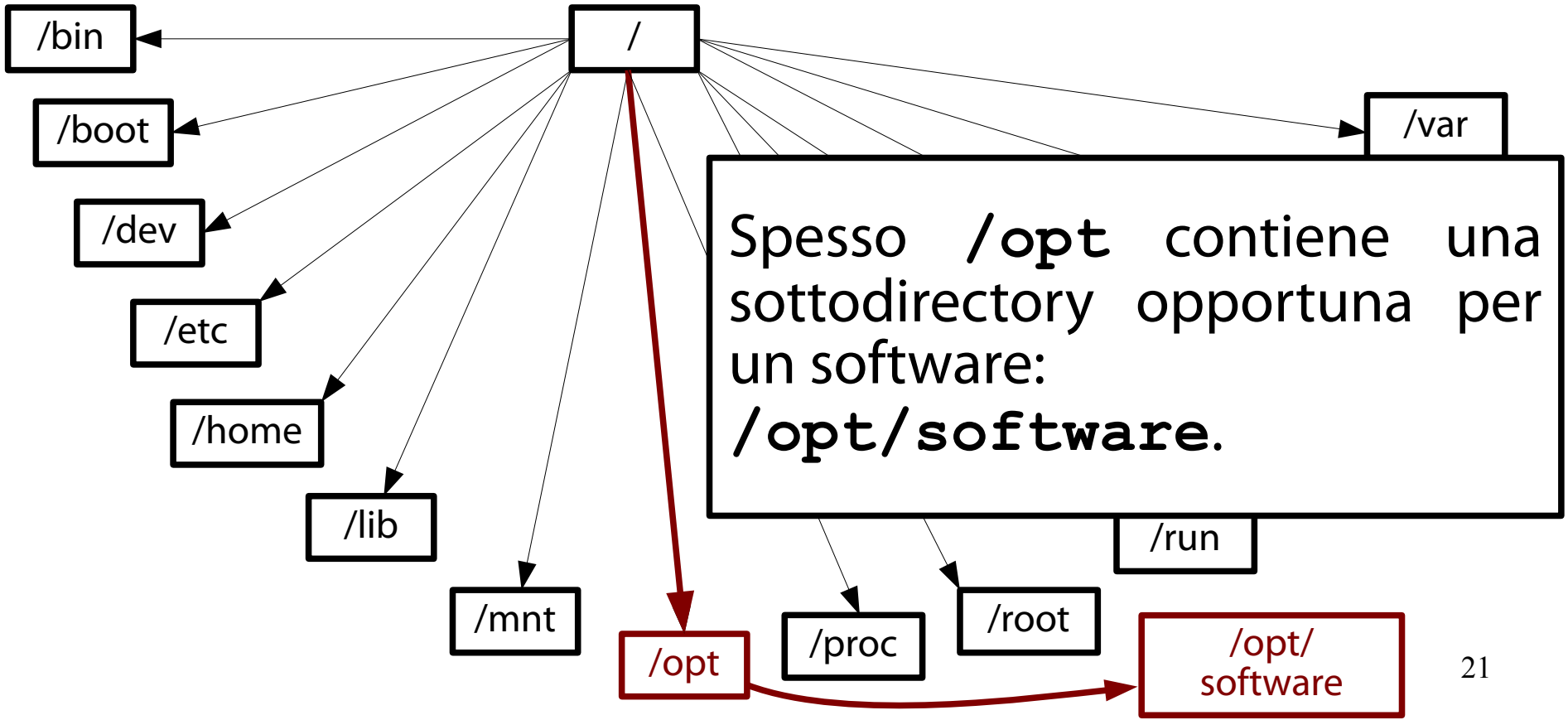
(Spesso presenti solo in forma binaria)



La directory **`/opt`** contiene un file system completo (con directory **`bin`**, **`sbin`**, **`lib`**, **`usr`**, …) destinato a software di terze parti non free e binario.

Applicazioni proprietarie third party

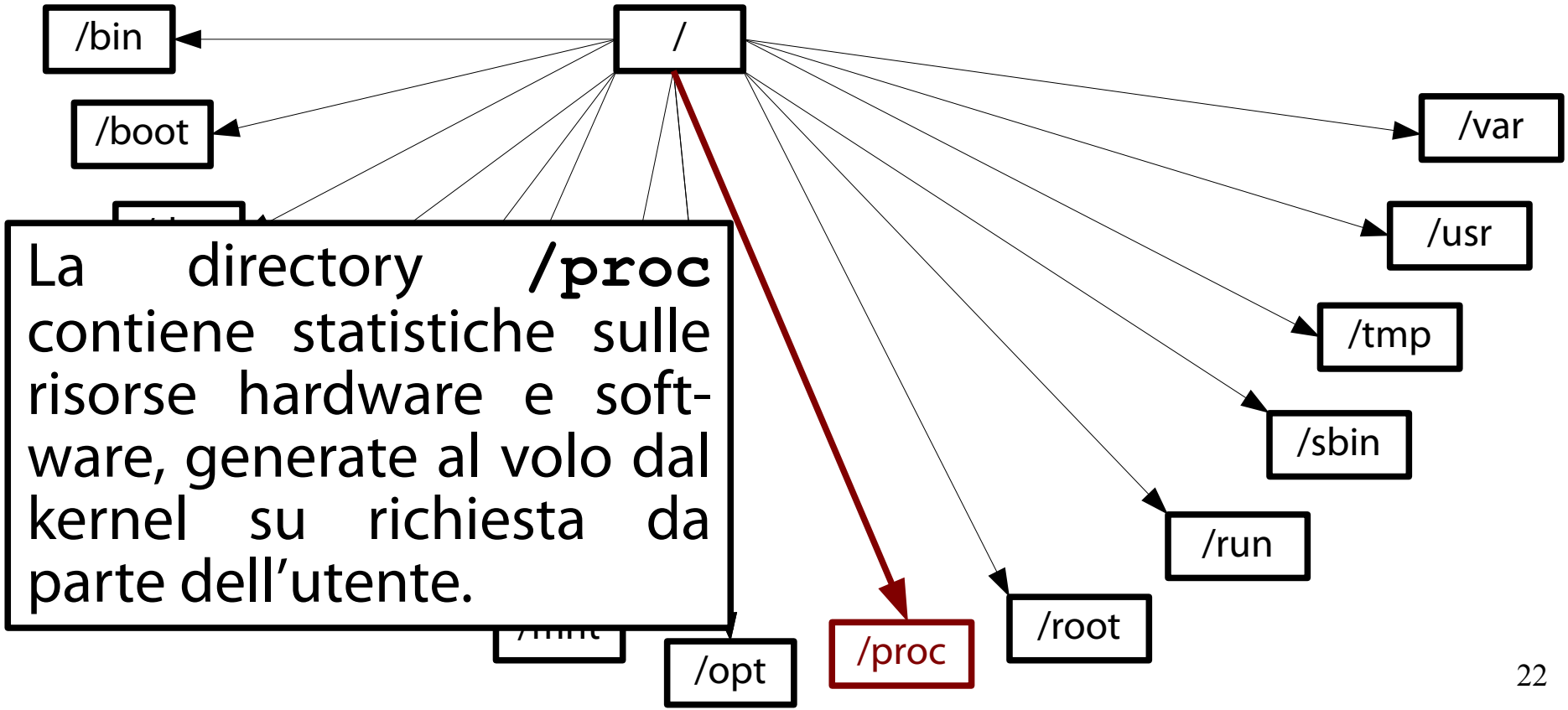
(Spesso presenti solo in forma binaria)



Spesso **`/opt`** contiene una sottodirectory opportuna per un software: **`/opt/software`**.

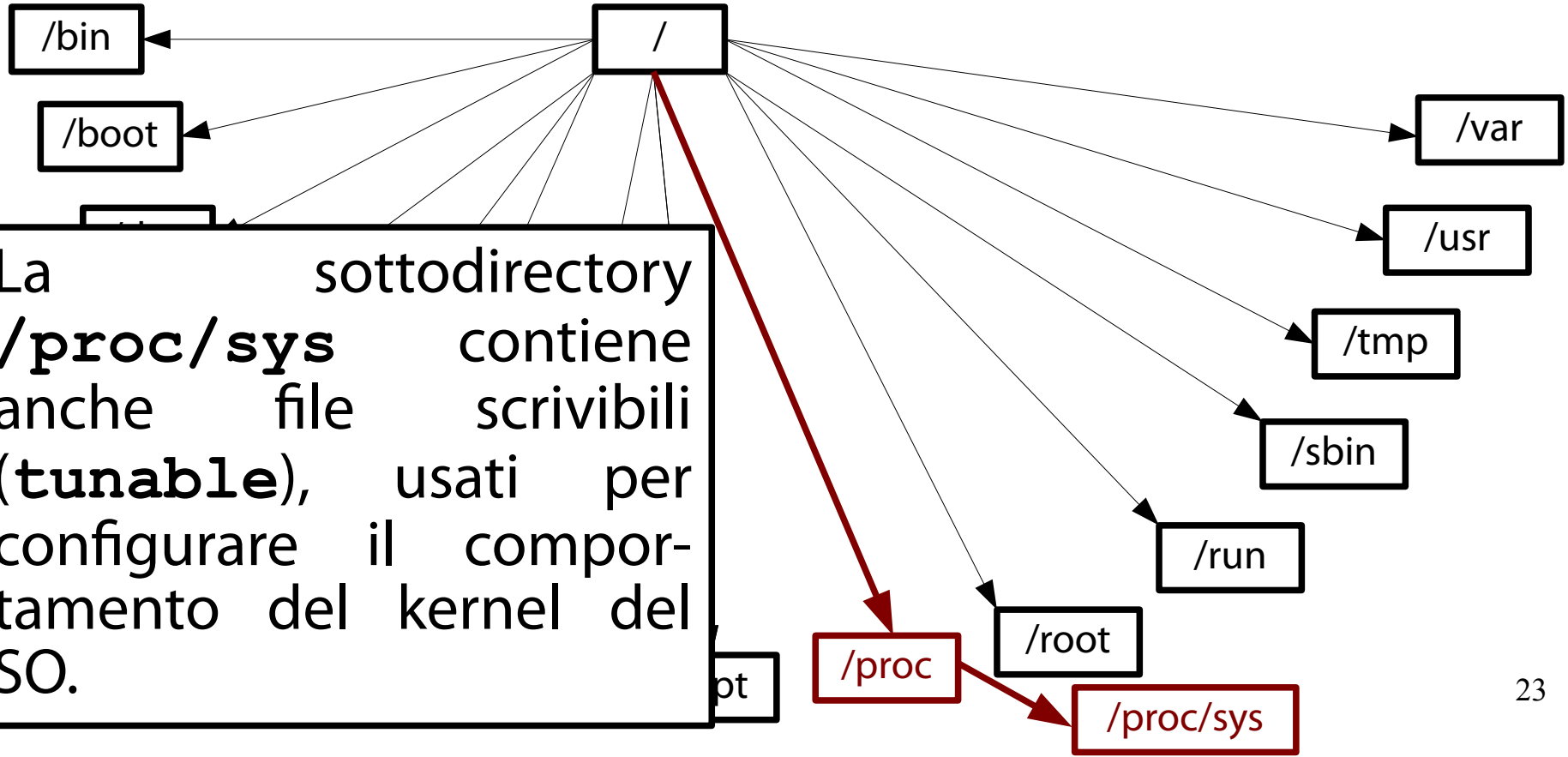
Statistiche fornite dal kernel

(Per le applicazioni di monitoraggio)



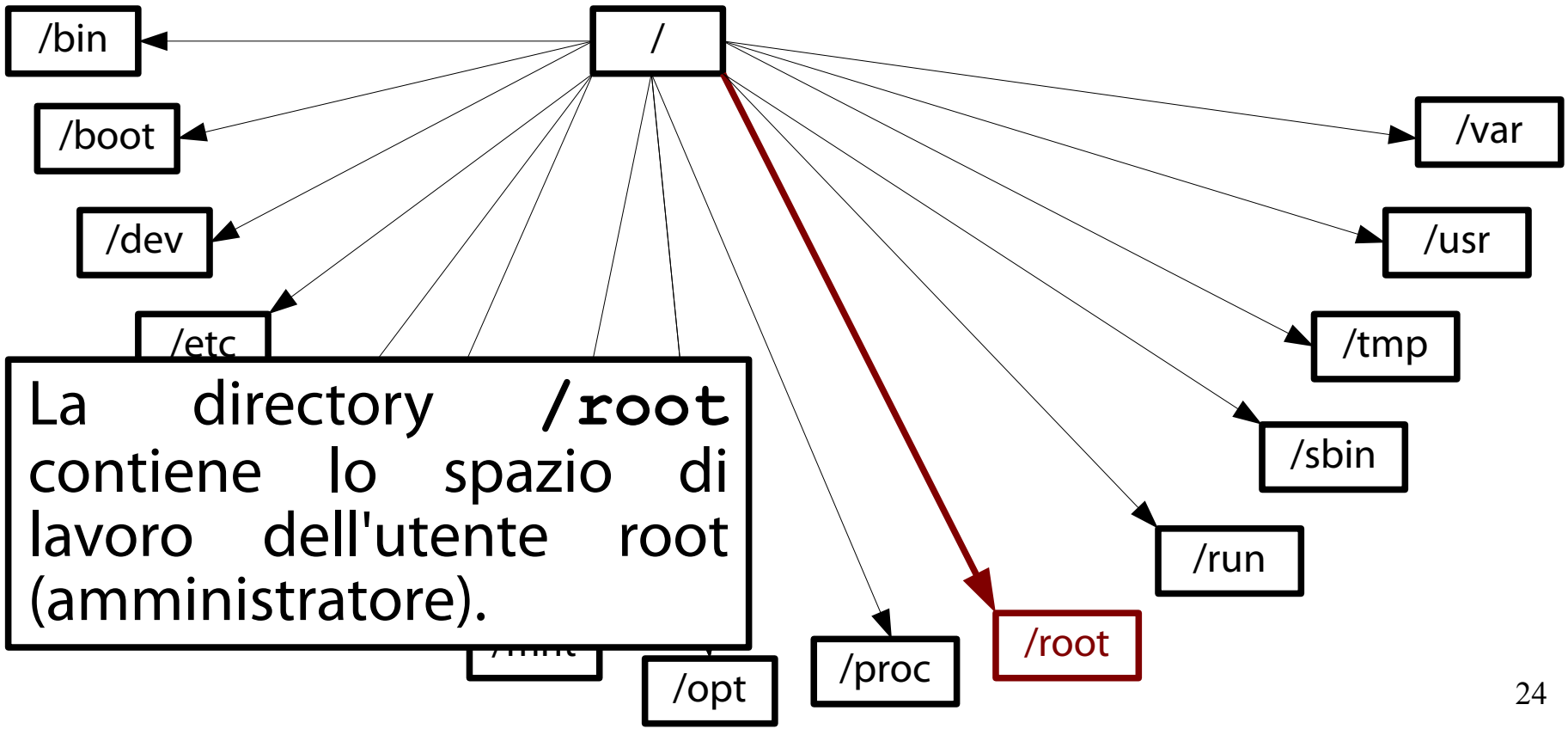
Tunable forniti dal kernel

(Per la configurazione di sottosistemi specifici)



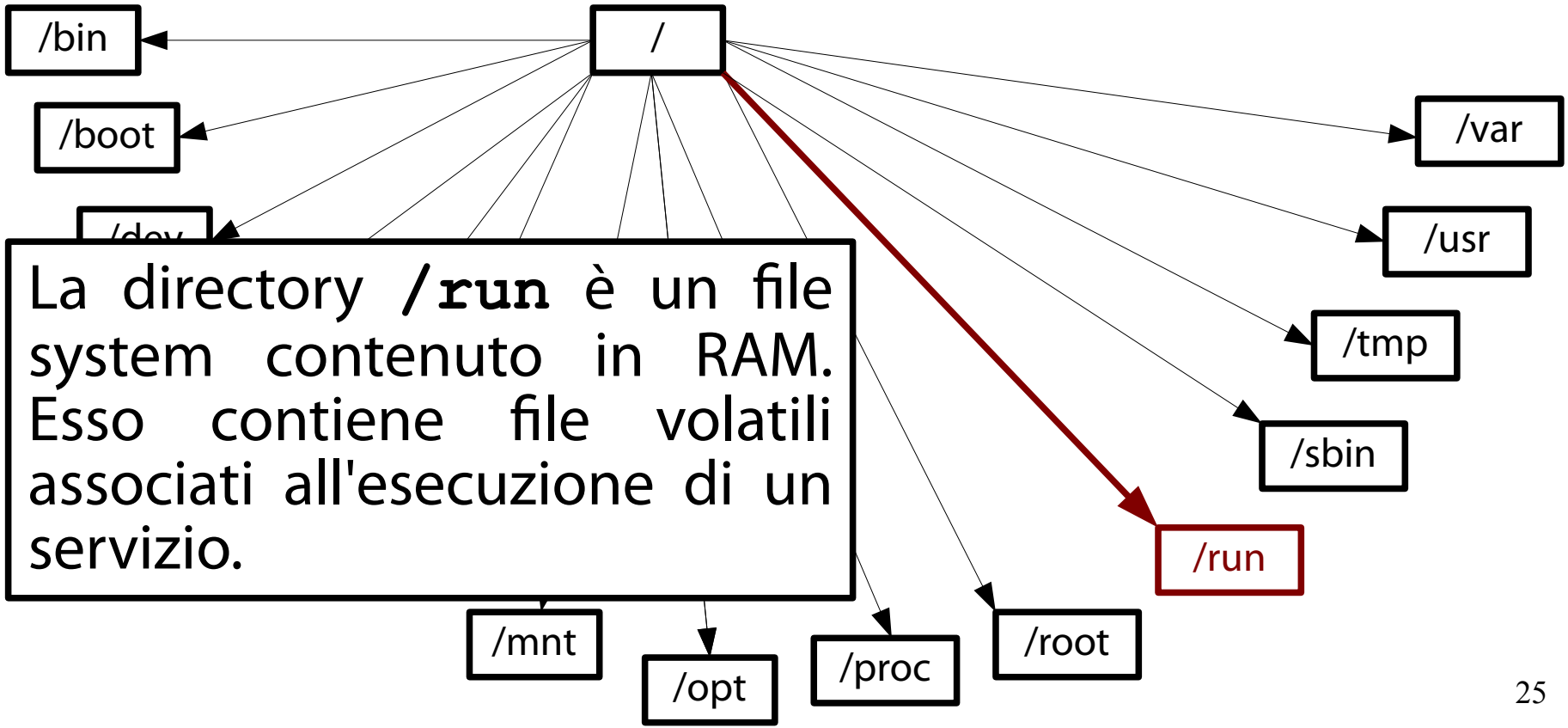
Home amministratore

(Del tutto equivalente alle altre home)



Dati volatili associati alle applicazioni

(Una volta terminate le applicazioni, non servono più e sono scartati)



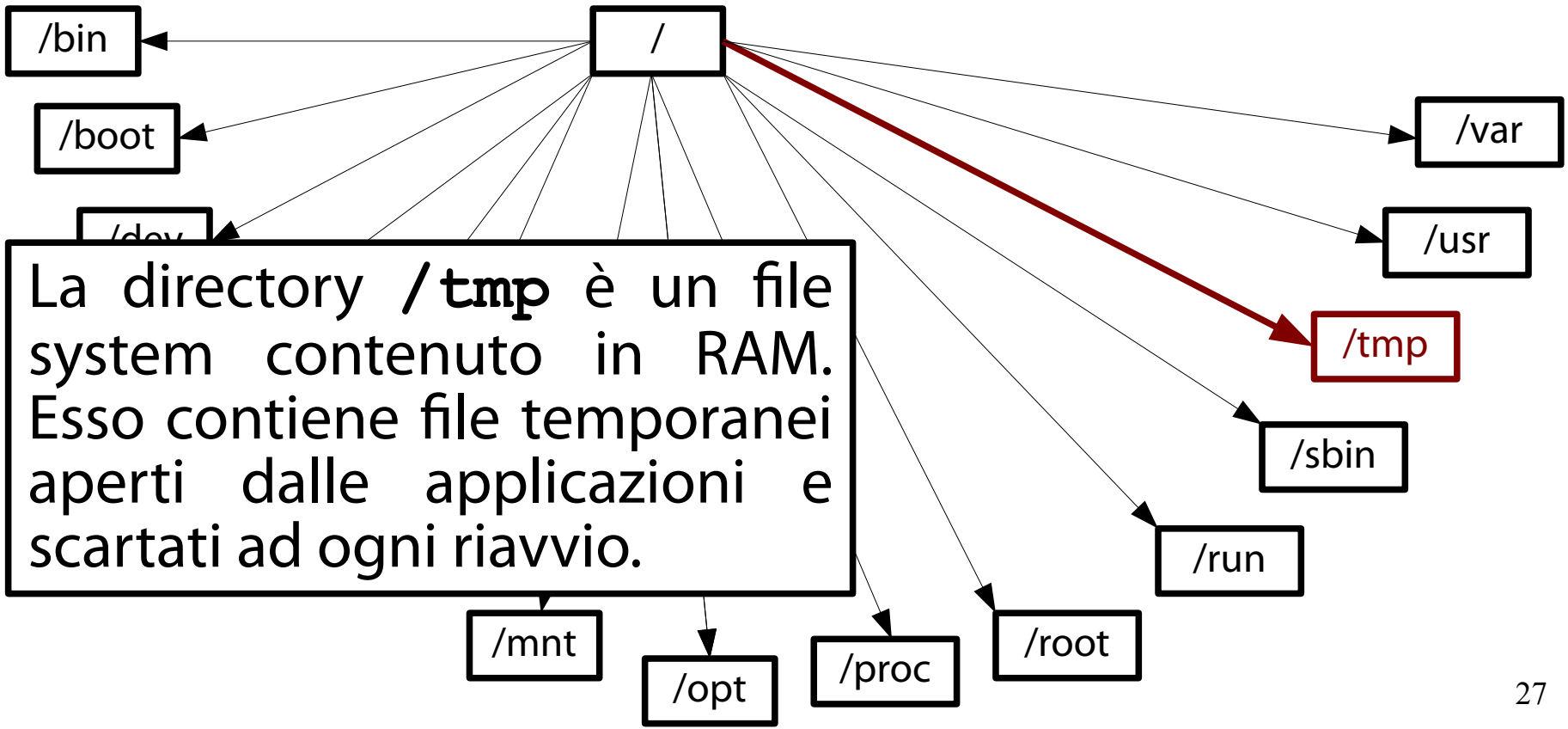
File volatili associati all'esecuzione?

(Eh?)

Ad esempio, gli script di avvio e terminazione di Apache2 (un Web server) devono sapere se il server è attivo oppure no e, nel caso, qual è l'identificatore univoco associato all'applicazione (Process IDentifier, PID).
Tali script gestiscono un file contenente il PID dell'applicazione: **`/run/http/httpd.pid`**.
Quando viene eseguito lo script di terminazione, viene letto il PID in tale file e terminata l'applicazione server.

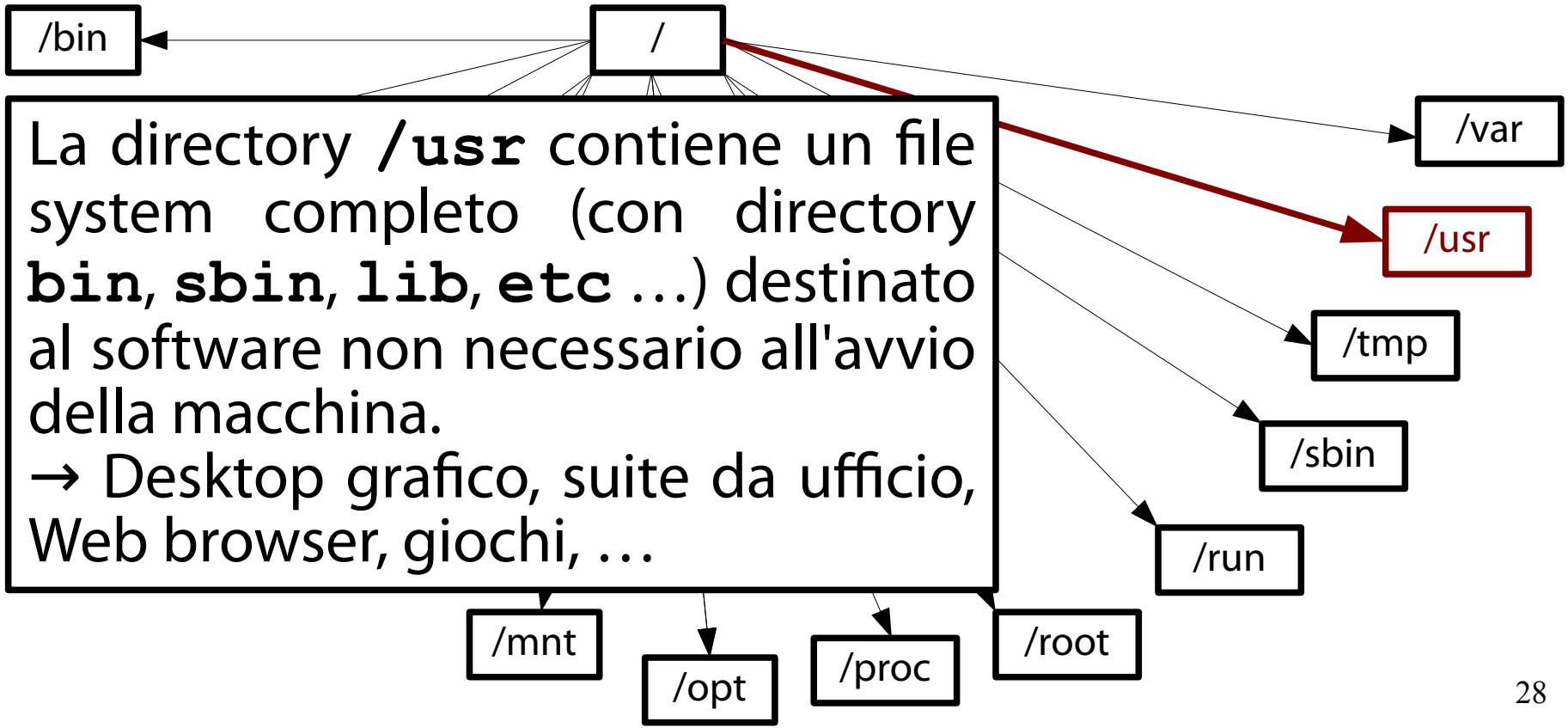
Directory temporanea

(Usata per contenere i file temporanei delle applicazioni; è accessibile a tutti)



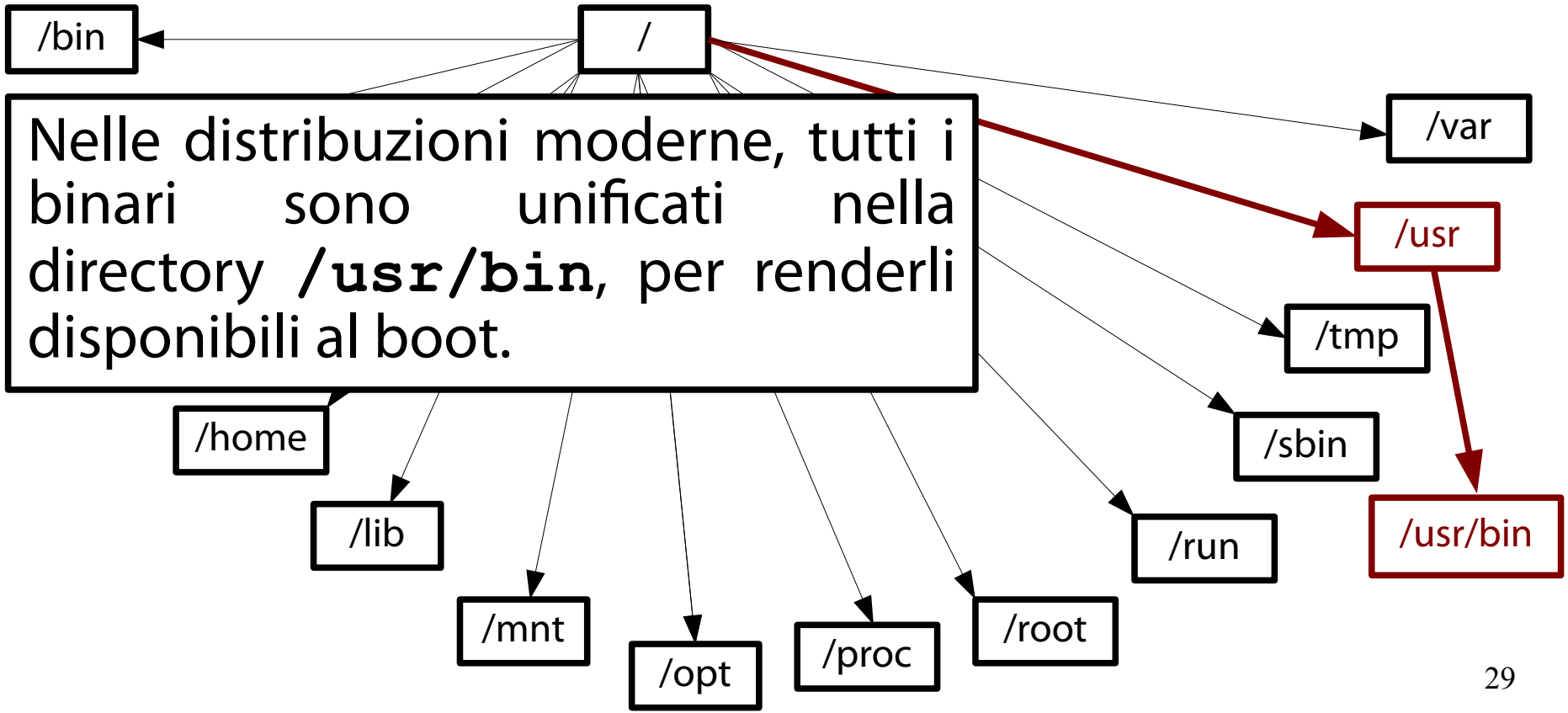
Applicazioni utente

(Si appoggiano al SO di base e forniscono un ambiente più ricco)



Unificazione dei binari in **`/usr/bin`**

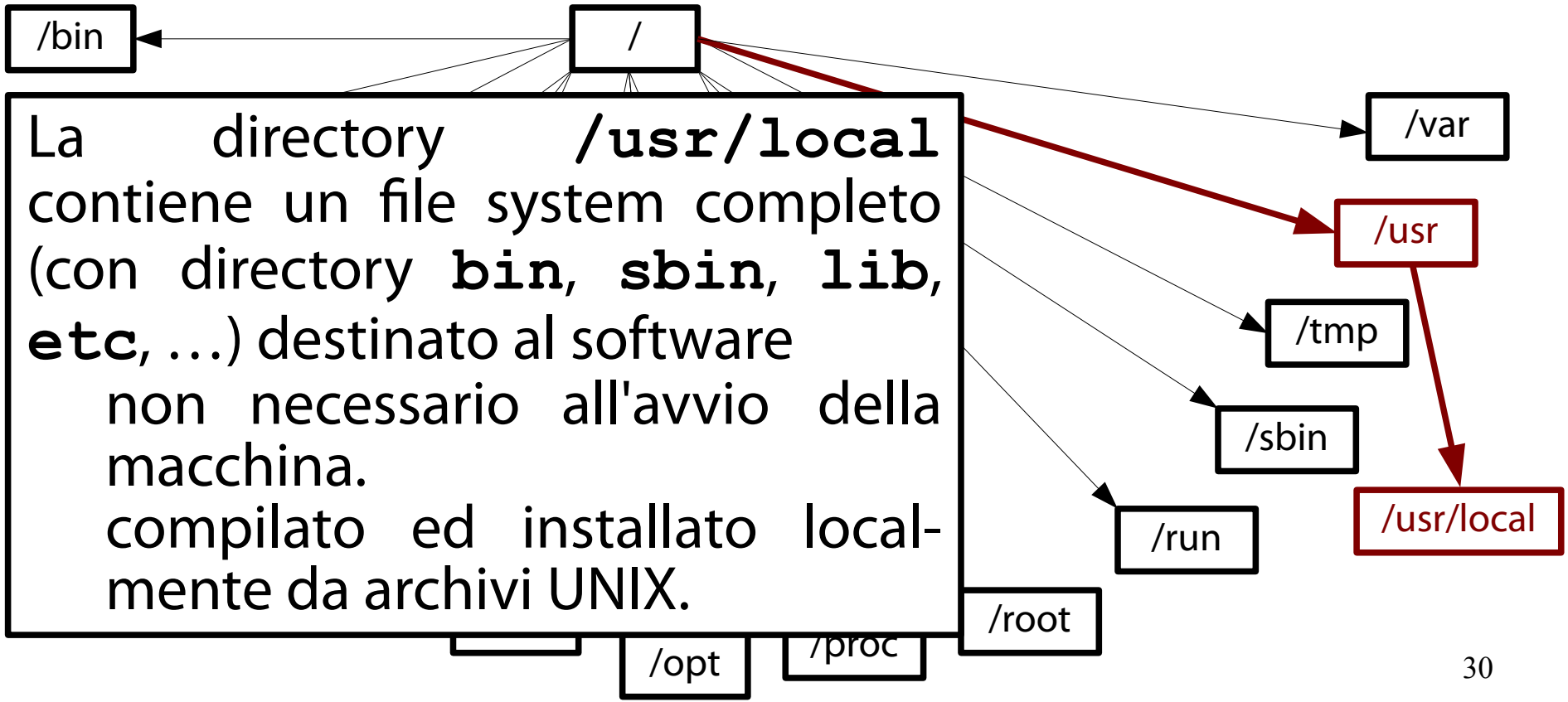
(Si evita lo sparpagliamento dei binari in quattro directory distinte)



Nelle distribuzioni moderne, tutti i binari sono unificati nella directory **`/usr/bin`**, per renderli disponibili al boot.

Applicazioni installate a mano

(Tipicamente, mediante gli strumenti GNU **`autotools`** e **`make`**)

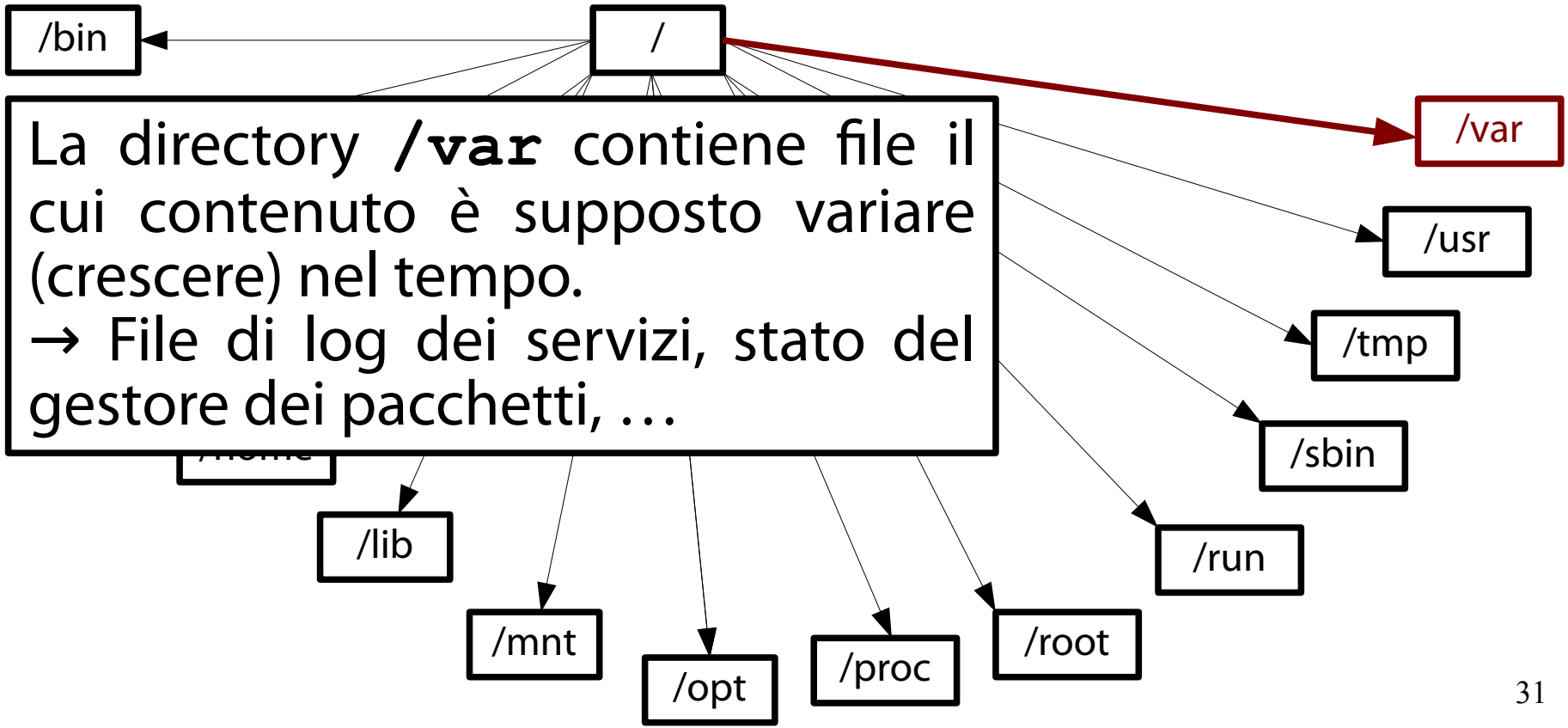


La directory **`/usr/local`** contiene un file system completo (con directory **`bin`**, **`sbin`**, **`lib`**, **`etc`**, …) destinato al software

- non necessario all'avvio della macchina.
- compilato ed installato local-mente da archivi UNIX.

File variabili

(File di log, cache di sistema, stato interno del gestore dei pacchetti, …)



Esercizio 1 (5 min.)

A quale file speciale di dispositivo corrisponde il dispositivo “primo disco SATA”? Che cosa si può ottenere con l’accesso a tale dispositivo?

GESTIONE DEI METADATI

Scenario e interrogativi

(Quali strumenti sono disponibili per la gestione dei metadati?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione per ottenere e modificare i metadati di file e directory.

Interrogativi:

Quali sono tali strumenti? Come funzionano?
Gli strumenti per file e directory sono simili?

Lettura dei metadati

(Comando stat; vale per file regolari, speciali e directory)

Il comando **`stat`** visualizza i metadati di un file. Il suo uso è semplicissimo:

```
stat [opzioni] nome_file
```

`stat` funziona con tutti i tipi di file:

- file regolari;
- directory;
- file speciali (di dispositivo, link simbolici, socket …).

Un esempio d'uso

(Su un file regolare)

Ad esempio, si provi a visualizzare l'insieme dei metadati del file **`/etc/profile`**:

`stat /etc/profile`

Si dovrebbe ottenere un output simile al seguente.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

File: il nome del file passato come argomento da linea di comando.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Dim.: la dimensione del file in byte.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Blocchi: il numero di blocchi allocati sul file system.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Blocco di IO: la dimensione di un blocco del file system in byte.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso
(Su un file regolare)

La tipologia del file.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Device: un identificatore unico del dispositivo (fisico o virtuale) che ospita il file system.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Device: un identificatore unico del dispositivo (fisico o virtuale) che ospita il file system.
Major number (**08**) e minor number (**01**) in esadecimale.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Device: un identificatore unico del dispositivo (fisico o virtuale) che ospita il file system.
Rappresentazione decimale (**2049**) di **801**.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761            Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d     Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Inode: un identificatore unico che punta alla struttura di controllo del file (**inode**), usata per rintracciarlo sul dispositivo (fisico o virtuale) che ospita il file system.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Coll.: il numero di collegamenti fisici al file.
In seguito se ne discuterà il significato.
Di default è pari ad 1.
Può aumentare se si crea un collegamento fisico (hard link) al file con il comando **`ln`**.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Accesso: i permessi di accesso al file (simbolici ed ottali).

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Uid: lo user ID ed il nome utente del creatore del file.
È usato per identificare in maniera univoca il creatore del file.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Gid: il group ID ed il nome del gruppo di lavoro del file. È usato per identificare in maniera univoca un insieme di utenti che condividono permessi di accesso al file.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Accesso: timestamp di ultimo accesso al file.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Modifica: timestamp di ultima modifica dei dati del file.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:48:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Cambio: timestamp di ultima modifica dei metadati del file.

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Un esempio d'uso

(Su un file regolare)

Creazione: timestamp di creazione del file ("-" → non supportato dal file system).

```
andreoli@debian-sistemi-operativi-2:~$ stat /etc/profile
  File: "/etc/profile"
  Dim.: 761             Blocchi: 8          Blocco di IO: 4096   file regolare
Device: 801h/2049d      Inode: 783551      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Accesso  : 2016-12-13 07:47:28.033459494 +0100
Modifica : 2014-10-22 16:02:30.000000000 +0200
Cambio   : 2015-10-08 17:46:17.130791000 +0200
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Esercizio 2 (3 min.)

Si stampi l'insieme dei metadati per i file seguenti:

```
$HOME
/tmp/.X11-unix/X0
/dev/sda1
/dev/tty0
```

Notate delle differenze?

Visione dei contenuti di una directory

(Si usa il comando **`ls`**)

Il comando **`ls`** visualizza il contenuto di uno più file o directory **FD_1**, **FD_2**, …, **FD_N**. Il suo uso è semplicissimo:

`ls [opzioni] FD1 ... FD`$_N$

Le opzioni consentono di abilitare le più disparate modalità di visualizzazione:

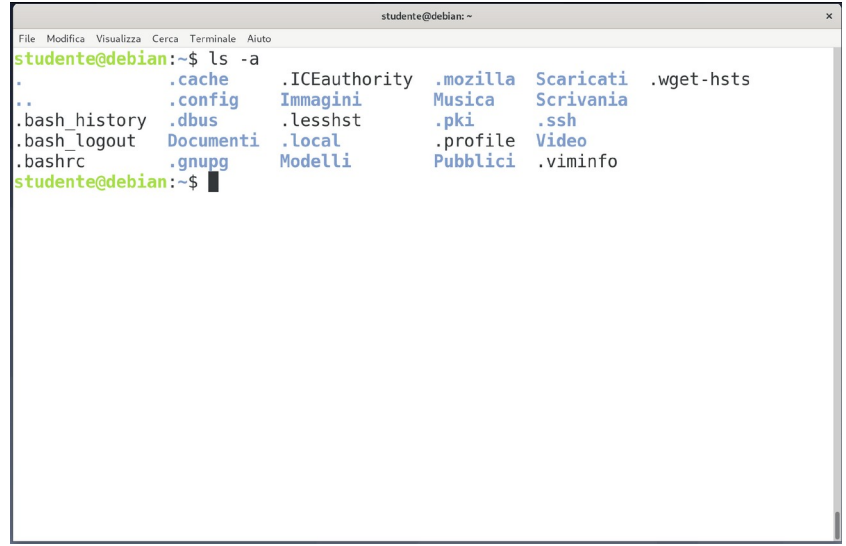
- ricorsiva;
- colorata;
- ordinata;
- file nascosti;
- …

Visualizzazione file nascosti

(Si usa l'opzione **`-a`** di **`ls`**)

L'opzione **`-a`** del comando **`ls`** abilita la visualizzazione dei file e delle directory nascoste (che iniziano con il carattere punto):

```
ls -a
```



Come si può vedere, sono sempre presenti almeno i due file nascosti seguenti:

- `.` → directory corrente;
- `..` → directory superiore.

Visualizzazione metadati

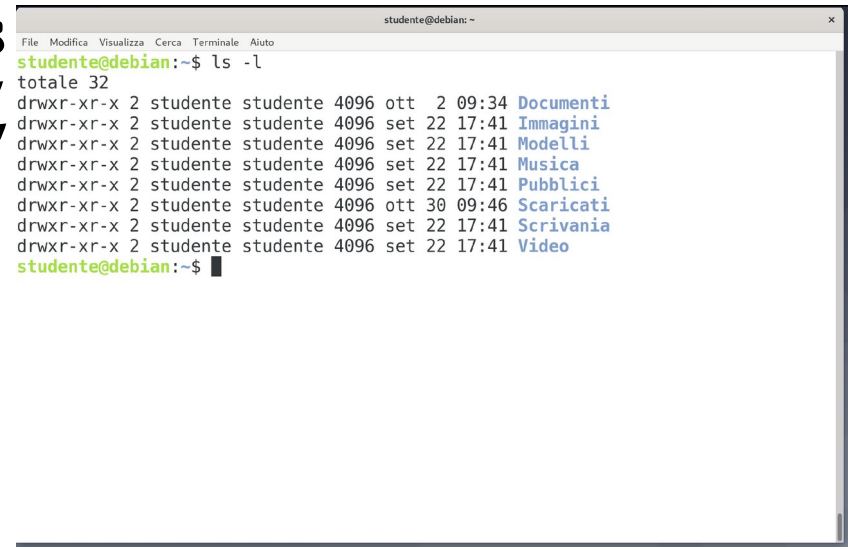
(Si usa l'opzione `-l` di `ls`)

L'opzione `-l` del comando `ls` abilita la visualizzazione "lunga", ovvero dei metadati principali:

```
ls -l
```

I metadati principali sono:

- tipo del file e permessi;
- numero di collegamenti fisici;
- utente creatore;
- gruppo di lavoro;
- dimensione (byte);
- tempo di accesso;
- nome.



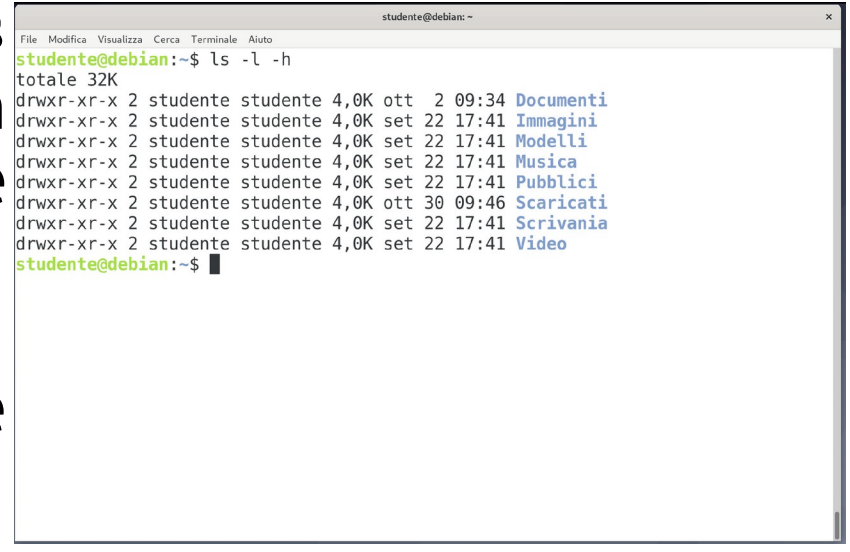
Visualizzazione dimensioni dei file

(Si usa l'opzione **`-h`** di **`ls`**)

L'opzione **`-h`** del comando **`ls`** stampa le dimensioni dei file in formato "umano", ovvero con le unità di misura internazionali:

```
ls -l -h
```

Si noti il cambio di dimensione 4096 → 4,0K.

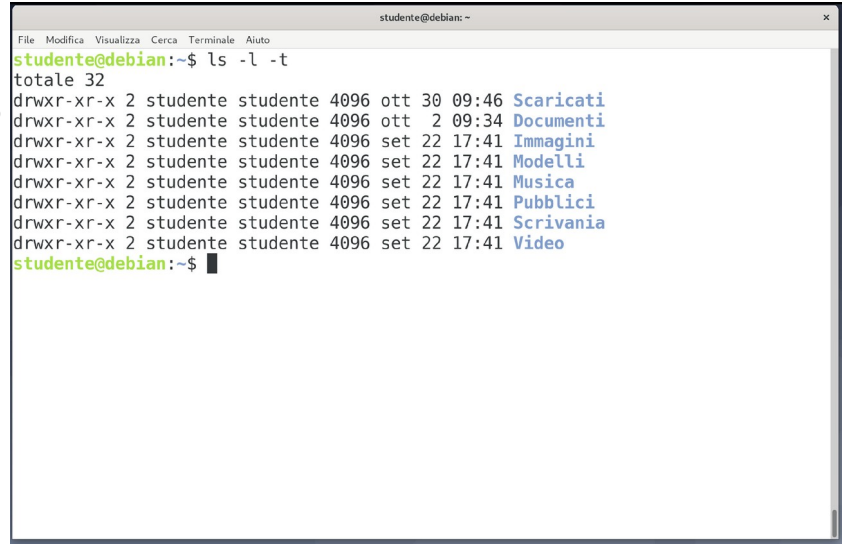


Ordinamento per tempo di accesso

(Si usa l'opzione **`-t`** di **`ls`**)

L'opzione **`-t`** del comando **`ls`** ordina i file per tempo di accesso decrescente:

```
ls -l -t
```

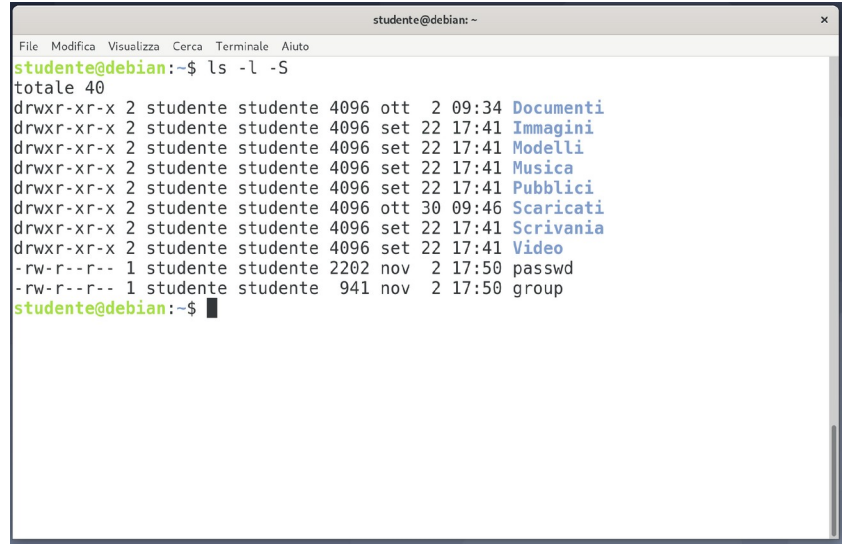


Ordinamento per dimensione

(Si usa l'opzione **`-S`** di **`ls`**)

L'opzione **`-S`** del comando **`ls`** ordina i file per dimensione decrescente:

```
ls -l -S
```

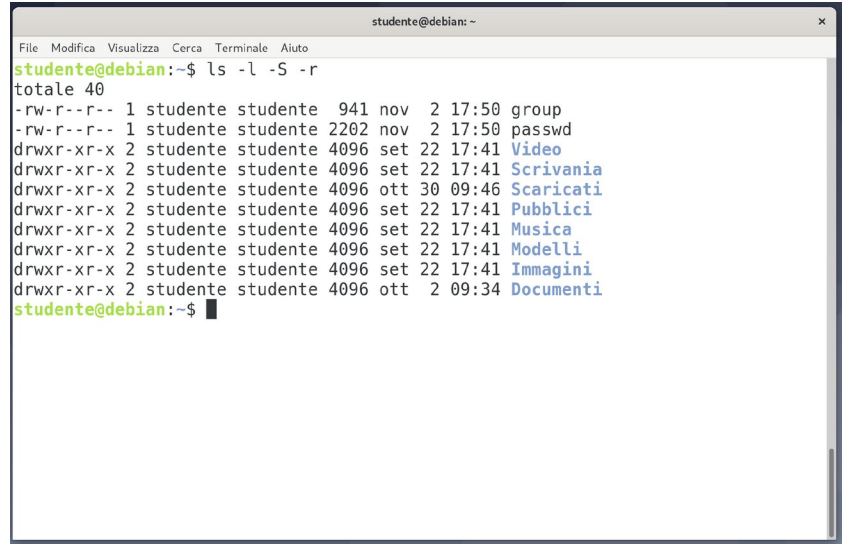


Inversione dell'ordinamento

(Si usa l'opzione **`-r`** di **`ls`**)

L'opzione **`-r`** del comando **`ls`** inverte l'ordinamento imposto con le opzioni **`-t`** e **`-S`**:

```
ls -l -S -r
```



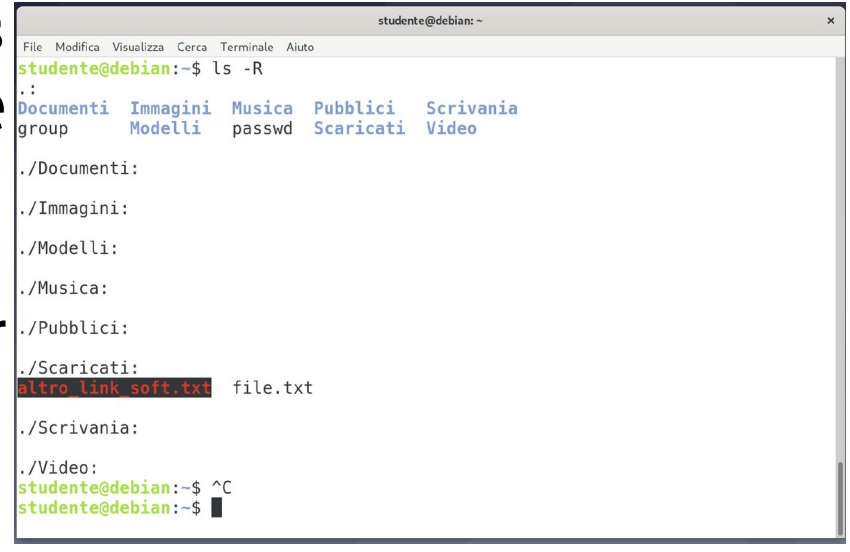
Elenco ricorsivo di file e directory

(Si usa l'opzione **`-S`** di **`ls`**)

L'opzione **`-R`** del comando **`ls`** elenca ricorsivamente i file e le sottodirectory:

```
ls -R
```

Il comando è stato interrotto per semplicità.



Esercizio 3 (1 min.)

Elencate la directory **`/etc`** nella modalità seguente:

- ricorsivamente;
- con i metadati;
- con i file nascosti.

Associazione di applicazioni di default

(Come individuare una applicazione che sappia gestire il contenuto di un file)

Si considerino i file di contenuto "regolare". Il contenuto di tali file acquista senso solo se gestito da una applicazione opportuna.

Si provi ad aprire un filmato MPEG con un editor di testo, tanto per intendersi…

Come fa il SO ad associare automaticamente una applicazione sensata ad un file regolare?

Filmato MPEG → Lettore multimediale.

File sorgente → Editor di testo da programmatore.

…

Associazione diretta

(Il SO decide l'applicazione di default)

Il meccanismo più semplice di riconoscimento del tipo di un file regolare è la **associazione diretta**.

Il SO legge una porzione del nome del file (estensione, sotto-stringa) ed associa automaticamente una applicazione tipo.

Il nome dell'applicazione tipo:

- è cablato nel SO (MS-DOS, UNIX linea di comando con il comando **`lesspipe`**).
- è scritto come metadato del file (Mac OS).

Lista di estensioni 1/2

(L'applicazione dichiara la sua lista di estensioni)

Ciascuna applicazione gestisce una lista di estensioni gradite.

Ad esempio, Libreoffice → .doc, .odt, .docx, ...

Diverse applicazioni possono leggere lo stesso tipo di file.

Abiword, Openoffice, Libreoffice → .doc

Una sola applicazione può essere impostata come il default per un dato tipo di file.

Lista di estensioni 2/2

(L'applicazione dichiara la sua lista di estensioni)

Quando un utente clicca due volte sull'icona di un file in un ambiente desktop, il SO carica l'applicazione di default.

Quando un utente clicca col tasto destro sull'icona del file e seleziona “Apri con…”, viene mostrata la lista delle applicazioni in grado di gestire il tipo di file.

Approccio usato nei SO con desktop grafico:

Windows, UNIX, MacOS.

Analisi del contenuto 1/2

(Approccio "signature-based", simile al modus operandi di un antivirus)

Ciascun file è riconoscibile da una o più sequenze di byte (dette **magic number**) in offset strategici.

Esempio: eseguibile UNIX in formato "ELF".

Contiene i caratteri 'E', 'L', 'F' nel secondo, terzo e quarto byte. Si digiti `less /bin/ls` per verificare.

Il SO contiene tutti i magic number in un database locale su file, detto **magic file**.

Analisi del contenuto 2/2

(Approccio "signature-based", simile al modus operandi di un antivirus)

Il SO individua il tipo di file:

verificando uno per uno tutti i gruppi di sequenze nel magic file;

fermandosi al primo gruppo di sequenze presente nel file.

Approccio usato nei sistemi UNIX.

Identificazione del tipo di file

(Si usa il comando **`file`**)

Il comando **`file`** stampa la tipologia di uno o più file o directory **FD_1**, **FD_2**, …, **FD_N**:

```
file FD1 FD2 ... FDN
```

Ad esempio, per scoprire il tipo di file di **`/etc/passwd`**, si esegue il comando seguente:

```
file /etc/passwd
```

Si ottiene l'output seguente:

```
/etc/passwd: ASCII text
```

Esercizio 4 (1 min.)

Individuate il tipo di file dei file seguenti:

```
$HOME
/tmp/.X11-unix/X0
/dev/sda1
/dev/tty0
/usr/bin/editor
```

CREAZIONE E RIMOZIONE

Scenario e interrogativi

(Quali strumenti sono disponibili per creare e rimuovere file e directory?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione per creare e rimuovere file e directory.

Interrogativi:

Quali sono tali strumenti? Come funzionano?
Gli strumenti per file e directory sono simili?

Creazione di file

(Fra i diversi modi possibili si annovera il comando **`touch`**)

Tramite la linea di comando è possibile creare un nuovo file nei modi seguenti.

- Invocazione di un editor e salvataggio di un file.
- Redirezione di STDOUT e/o STDERR (discussa in seguito, nella sezione dedicata a BASH).
- Il comando **`touch`**.

È a quest'ultimo comando che ci si riferisce ora.

Il comando **touch**

(**Crea file vuoti**; altera i timestamp del file)

Il comando **touch** nasce per modificare i timestamp associati ad un file.

Nella sua accezione più semplice, **touch** è lanciato senza opzioni e con un elenco di nomi di file non esistenti **F1**, **F2**, … **FN**. In tali condizioni, **touch** crea file con contenuto nullo e timestamp riferito all'istante attuale:

```
touch F1 F2 ... FN
```

Il comando `touch`

(**Crea file vuoti**; altera i timestamp del file)

Ad esempio, per creare un nuovo file vuoto dal nome **`file_di_prova`**, si esegue il comando seguente:

```
touch file_di_prova
```

Si esaminano i metadati del file con il comando seguente:

```
stat file_di_prova
```

I metadati del nuovo file

(File di dimensione nulla; timestamp identici a pari all'istante di creazione)

Dimensione nulla

```
andreoli@debian-sistemi-operativi-2:~$ stat file_di_prova
  File: "file_di_prova"
  Dim.: 0               Blocchi: 0          Blocco di IO: 4096   file regolare vuoto
Device: 801h/2049d      Inode: 263088      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: ( 1000/andreoli)   Gid: ( 1000/andreoli)
Accesso  : 2016-12-14 21:20:47.344122595 +0100
Modifica : 2016-12-14 21:20:47.344122595 +0100
Cambio   : 2016-12-14 21:20:47.344122595 +0100
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Timestamp identici

Il comando `touch`

(Crea file vuoti; **altera i timestamp del file**)

Lanciato con l'opzione **`-a`** su un file esistente, il comando **`touch`** ne altera il timestamp di ultimo accesso.
Si attenda un minuto e si esegua il comando seguente:

```
touch -a file_di_prova
```

Il comando `touch`

(Crea file vuoti; **altera i timestamp del file**)

Lanciato con l'opzione **`-m`** su un file esistente, il comando **`touch`** ne altera il timestamp di ultima modifica dei dati. Si attenda un minuto e si esegua il comando seguente:

```
touch -m file_di_prova
```

I metadati del nuovo file

(File di dimensione nulla; timestamp identici a pari all'istante di creazione)

Si esaminino i metadati del file:

`stat file_di_prova`

```
andreoli@debian-sistemi-operativi-2:~$ stat file_di_prova
  File: "file_di_prova"
  Dim.: 0               Blocchi: 0          Blocco di IO: 4096   file regolare vuoto
Device: 801h/2049d      Inode: 263088      Coll.: 1
Accesso: (0644/-rw-r--r--)  Uid: ( 1000/andreoli)   Gid: ( 1000/andreoli)
Accesso  : 2016-12-14 21:32:21.964153152 +0100
Modifica : 2016-12-14 21:33:11.476155330 +0100
Cambio   : 2016-12-14 21:33:11.476155330 +0100
Creazione: -
andreoli@debian-sistemi-operativi-2:~$
```

Il timestamp di ultima modifica è più in avanti di un minuto rispetto al timestamp di ultimo accesso

Poiché i timestamp sono metadati, il timestamp di ultimo cambio metadati è identico al timestamp di ultima modifica dei dati

Esercizio 5 (3 min.)

Si crei un file nuovo di contenuto nullo. Leggendo la pagina manuale del comando relativo, si individui un modo per impostare i timestamp al 1/1/1970, ore 00:00.

Creazione di directory

(Si usa il comando **mkdir**)

Il comando **mkdir** crea directory vuote. Nella sua accezione più semplice, **mkdir** è lanciato senza opzioni e con un elenco di nomi di directory non esistenti **D1**, **D2**, … **DN**:

```
mkdir D1 D2 ... DN
```

Ad esempio, per creare una directory **dir_di_prova** si esegue il comando comando seguente:

```
mkdir dir_di_prova
```

Problemi legati a **`mkdir`** 1/2

(**`mkdir`** fallisce se si prova a creare una directory già esistente)

Il comando **`mkdir`** fallisce se si prova a creare una directory già esistente.

Ad esempio, si provi a creare la directory **`a`**:

```
mkdir a
```

Si provi a creare nuovamente la directory **`a`**:

```
mkdir a
```

Si dovrebbe ottenere il messaggio di errore seguente:

```
mkdir: impossibile creare la directory
"a": File già esistente
```

Problemi legati a **`mkdir`** 2/2

(**`mkdir`** fallisce se manca una delle directory superiori)

Il comando **`mkdir`** fallisce se si prova a creare una sottodirectory senza aver prima creato le directory superiori.

Ad esempio, si crei la directory **`b`**:

```
mkdir b
```

Si crei la sottodirectory **`c/d`** di **`b`**, senza creare **`c`**:

```
mkdir b/c/d
```

Si dovrebbe ottenere il messaggio di errore seguente:

```
mkdir: impossibile creare la directory
"b/c/d": File o directory non esistente
```

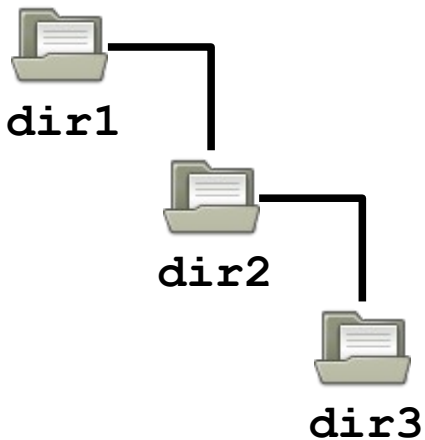
Creazione ricorsiva di sottodirectory

(Si usa l'opzione **`-p`** di **`mkdir`**)

L'opzione **`-p`** di **`mkdir`** risolve e i due problemi ora visti.
Se la directory è già esistente, non viene segnalato alcun errore.
Se manca una directory, viene creata.

Esercizio 6 (1 min.)

Si crei la seguente gerarchia di directory.



Cancellazione di file

(Si usa il comando **rm**)

Il comando **rm** cancella file e directory. Nella sua accezione più semplice, **rm** è lanciato senza opzioni e con un elenco di nomi di file esistenti **F1**, **F2**, … **FN**:

```
rm F1 F2 ... FN
```

Ad esempio, per cancellare il file **file_di_prova** si esegue il comando seguente:

```
rm file_di_prova
```

Si esaminino i metadati del file:

```
stat file_di_prova
```

Il file è sparito.

Cancellazione di directory

(Non immediatamente fattibile con **`rm`**)

Se si prova a cancellare una directory non vuota, **`rm`** si lamenta del fatto che **`dir1`** non sia vuota:

```
rm dir1
```

Purtroppo **`rm`** si rifiuta di cancellare anche directory vuote. Provare per credere:

```
mkdir a
rm a
```

Cancellazione di directory vuote

(Si usa l'opzione **`-d`** di **`rm`**)

Per poter rimuovere una directory vuota, si usa l'opzione **`-d`**.

Ad esempio, per cancellare la directory **`a`** si esegue il comando seguente:

```
rm -d a
```

Si esaminino i metadati della directory:

```
stat a
```

La directory è sparita.

Cancellazione ricorsiva di directory

(Si usa l'opzione **`-r`** di **`rm`**)

L'opzione **`-r`** di **`rm`** abilita la modalità di cancellazione ricorsiva. Se uno dei parametri è una directory, viene rimossa la directory ed il suo contenuto.

Ad esempio, per cancellare la gerarchia di directory creata nell'Esercizio 4, si esegue il comando seguente:

```
rm -r dir1
```

Si esaminino i metadati della directory:

```
stat dir1
```

La directory è sparita (e con essa, **`dir2`** e **`dir3`**).

Cancellazione interattiva

(Si usa l'opzione **`-i`** di **`rm`**)

L'opzione **`-i`** abilita la modalità interattiva. Prima di ogni cancellazione viene chiesta conferma all'utente.
Alcune distribuzioni GNU/Linux introducono un alias di **`rm`** a **`rm -i`** come misura protettiva contro disastri accidentali.

Ad esempio, si può creare un file vuoto:

```
touch file_di_prova
```

Si provi a cancellare il file in modalità interattiva.

```
rm -i file_di_prova
```

Si confermi con le stringhe **`y`**, **`yes`**, **`s`** o **`si`**.

Cancellazione forzata

(Si usa l'opzione **`-f`** di **`rm`**)

L'opzione **`-f`** abilita la modalità forzata. Si ignorano file e directory non esistenti; non viene chiesta alcuna conferma all'utente.
Questa modalità si usa per tipicamente per scavalcare la modalità interattiva introdotta in un alias.

Ad esempio, si può creare un file vuoto:

```
touch file_di_prova
```

Si provi a cancellare il file in modalità forzata.

```
rm -f file_di_prova
```

Cancellazione di directory

(Si può anche usare il comando **`rmdir`**)

Esiste anche il comando **`rmdir`**, duale di **`mkdir`**. Nella sua accezione più semplice, **`rmdir`** è lanciato senza opzioni e con un elenco di nomi di directory esistenti **`D1`**, **`D2`**, … **`DN`**:

```
rmdir D1 D2 ... DN
```

Ad esempio, per rimuovere la directory **`a`** si esegue il comando seguente:

```
rmdir a
```

ATTENZIONE! **`rmdir`** non cancella directory piene!

Cancellazione di gerarchie di directory

(Si usa l'opzione **`-p`** di **`rmdir`**)

Analogamente al comando **`mkdir`**, l'opzione **`-p`** di **`rmdir`** consente di cancellare una gerarchia di sottodirectory (che devono, tuttavia, essere altrimenti vuote).

Ad esempio, si crei la gerarchia di directory seguente:

```
mkdir -p dir1/dir2/dir3
```

Si cancelli la gerarchia di directory con **`rmdir -p`**:

```
rmdir -p dir1/dir2/dir3
```

Esercizio 7 (2 min.)

Create dieci file vuoti con i nomi seguenti:

```
file1  file2  file3  file4  file5  file6
file7 file8 file9 file10
```

Cancellate i file appena creati.

COPIA E SPOSTAMENTO

Scenario e interrogativi

(Quali strumenti sono disponibili per copiare e spostare file e directory?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione copiare e spostare file e directory.

Interrogativi:

Quali sono tali strumenti? Come funzionano?
Gli strumenti per file e directory sono simili?

Copia di file

(Si usa il comando **cp**)

Il comando **cp** copia file e/o directory. Nella sua accezione più semplice, **cp** è lanciato senza opzioni e con due argomenti **FSRC**, **FDST**. In tali condizioni, **cp** copia il file **FSRC** nel file **FDST**:

```
cp FSRC FDST
```

Ad esempio, si crei un file vuoto:

```
touch nuovo_file
```

Si attenda un minuto e si digiti:

```
cp nuovo_file copia_nuovo_file
```

Copia di più file in una directory

(L'ultimo argomento è una directory, gli altri sono file)

Se i primi N argomenti sono file **F1**, **F2**, … **FN** e l'ultimo argomento è una directory **D**, **cp** copia tutti i file nella directory:

```
cp F1 F2 … FN D
```

Ad esempio, si creino due file vuoti:

```
touch file1 file2
```

Si crei una directory vuota:

```
mkdir dir1
```

Per copiare **file1** e **file2** in **dir1**, si esegue:

```
cp file1 file2 dir1
```

Copia archiviale di una directory

(Si usa l'opzione **-a** di **cp**)

L'opzione **-a** di **cp** abilità la modalità di copia archiviale. Se i primi N argomenti sono file o directory **FD1**, **FD2**, … **FDN** e l'ultimo argomento è una directory **D**, **cp** copia fedelmente (ovvero, cercando di preservare i metadati originali) tutti i file/directory **FDx** nella directory destinazione **D**:

```
cp -a FD1 FD2 … FDN D
```

Ad esempio, per copiare **/tmp** nella directory corrente:

```
cp -a /tmp .
```

Esercizio 8 (2 min.)

Create un backup della propria home directory nella directory **`/tmp`**.

Spostamento di file

(Si usa il comando **mv**)

Il comando **mv** sposta file e/o directory. Nella sua accezione più semplice, **mv** è lanciato senza opzioni e con due argomenti **FDSRC**, **FDDST**. In tali condizioni, **mv** sposta il file/directory **FDSRC** nel file/directory **FDDST**:

```
mv FDSRC FDDST
```

Ad esempio, si crei un file vuoto:

```
touch nuovo_file
```

Si sposti il file appena creato nella directory **/tmp**:

```
mv nuovo_file /tmp
```

Spostamento di file con cambio nome

(Si usa sempre il comando **mv**)

È possibile cambiare il nome del file contestualmente al suo spostamento. È sufficiente usare un nome diverso in **FDDST**.

Ad esempio, si crei un file vuoto:

```
touch nuovo_file
```

Si sposti il file appena creato nella directory **/tmp** con nome **nuovo_file_spostato**:

```
mv nuovo_file /tmp/nuovo_file_spostato
```

Esercizio 9 (3 min.)

Studiate la pagina di manuale di **`mv`** e individuate un metodo per creare backup dei file sovrascritti.

VISUALIZZAZIONE CONTENUTO

Scenario e interrogativi

(Quali strumenti sono disponibili per la visualizzazione del contenuto dei file?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione per la visualizzazione del contenuto dei file.

Interrogativi:

Quali sono tali strumenti? Come funzionano?
Quali visualizzazioni sono possibili?

Visualizzazione di un file

(Si usa il comando **cat**)

Il comando più immediato per la visualizzazione non interattiva di un file è **cat**. Nella sua accezione più semplice, **cat** è lanciato senza opzioni e con un elenco di nomi di file esistenti $\mathbf{F_1}$, $\mathbf{F_2}$, … $\mathbf{F_N}$. In tali condizioni, cat stampa il contenuto di $\mathbf{F_1}$, $\mathbf{F_2}$, … $\mathbf{F_N}$ in sequenza.

```
cat F1 F2 ... FN
```

Ad esempio, per visualizzare il contenuto del file **/etc/profile**, si esegue il comando seguente:

```
cat /etc/profile
```

Stampa caratteri non stampabili

(Si usa l'opzione **-E** di **cat** per stampare i fine linea)

L'opzione **-E** di **cat** permette di stampare carattere di fine linea con il simbolo **$**.

Ad esempio, per evidenziare i fine linea del file **/etc/passwd**, si esegue il comando seguente:

```
cat -E /etc/passwd
```

Si ottiene l'output seguente:

```
root:x:0:0::/root:/bin/bash$
bin:x:1:1::/:/sbin/nologin$
daemon:x:2:2::/:/sbin/nologin$
...
```

Stampa caratteri non stampabili

(Si usa l'opzione **`-T`** di **`cat`** per stampare i caratteri di tabulazione)

L'opzione **`-T`** di **`cat`** permette di stampare il carattere di tabulazione con il simbolo **`^I`**.

Ad esempio, per evidenziare i caratteri di tabulazione nel file **`/etc/skel/.bashrc`**, si esegue il comando seguente:

```
cat -T /etc/skel/.bashrc
```

L'output del comando

(Si nota qualcosa di strano?)

```
...
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
^I. "$HOME/.bashrc"
    fi
fi
...
```

Stampa caratteri non stampabili

(Si usa l'opzione **`-v`** di **`cat`** per stampare i caratteri non stampabili)

L'opzione **`-v`** di **`cat`** permette di stampare i caratteri altrimenti non stampabili. Quali sono questi caratteri?

Caratteri di controllo ASCII.

Hanno codice ASCII nell'intervallo [0, 31].

Controllano l'output del terminale.

Sono usabili da terminale tramite la sequenza di tasti **`<CTRL>-X`**, dove **`X`** è un carattere opportuno.

Sono stampati da **`cat -v`** con il simbolo **`^X`**, dove **`X`** è il carattere di cui sopra.

Stampa caratteri non stampabili

(Si usa l'opzione **`-v`** di **`cat`** per stampare i caratteri non stampabili)

L'opzione **`-v`** di **`cat`** permette di stampare i caratteri altrimenti non stampabili. Quali sono questi caratteri?

Caratteri non ASCII.

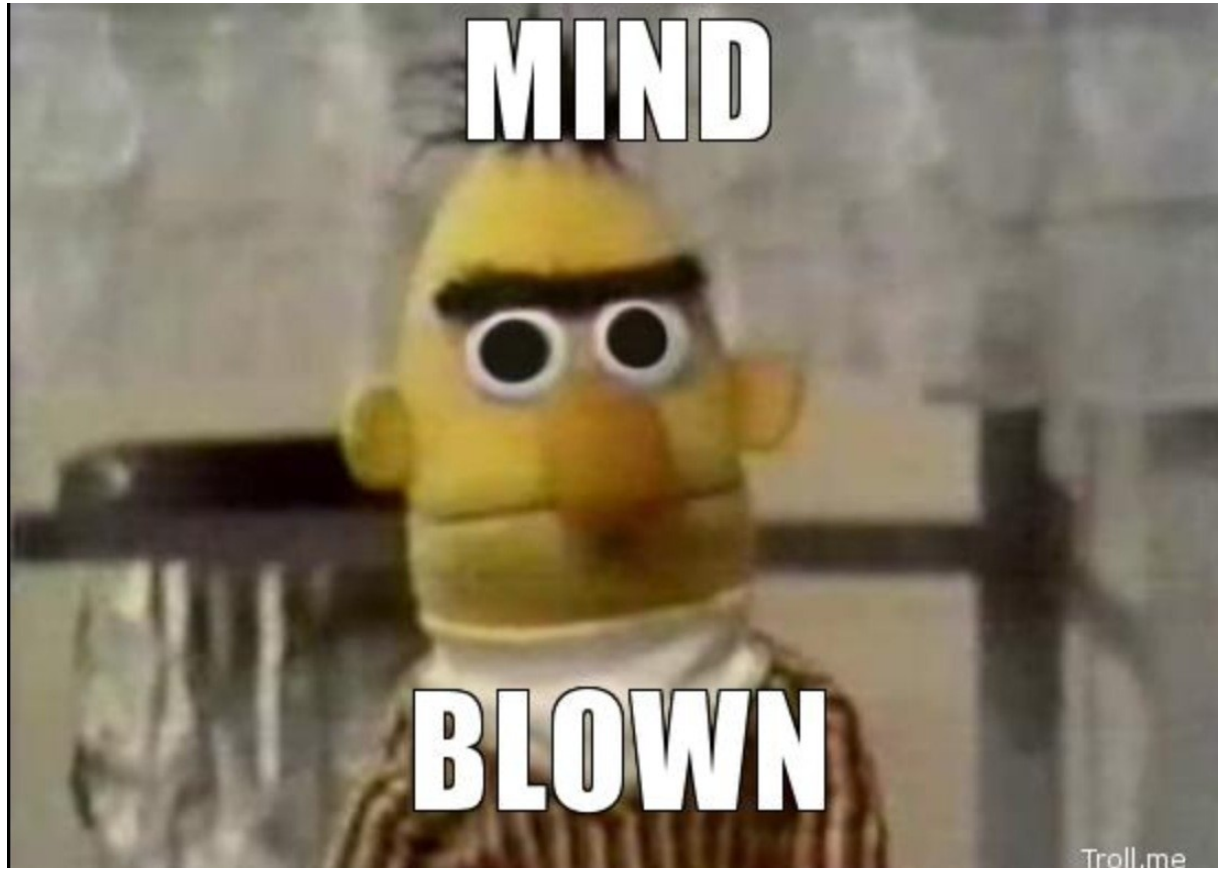
Caratteri codificati con una delle tante codifiche ASCII estese ISO 8859 (codifica a singolo byte, intervallo [128, 255]).

Caratteri codificati con la codifica universale UTF-8 (codifica multi byte, retrocompatibile con ASCII).

Sono stampati da **`cat -v`** con il simbolo **`M-X`** (dove **`X`** è un carattere opportuno) per ogni byte di codifica del carattere.

“Controllo”? “Extra”? “^”? “M-”? Eh?

(What the $#£%&! is that?)



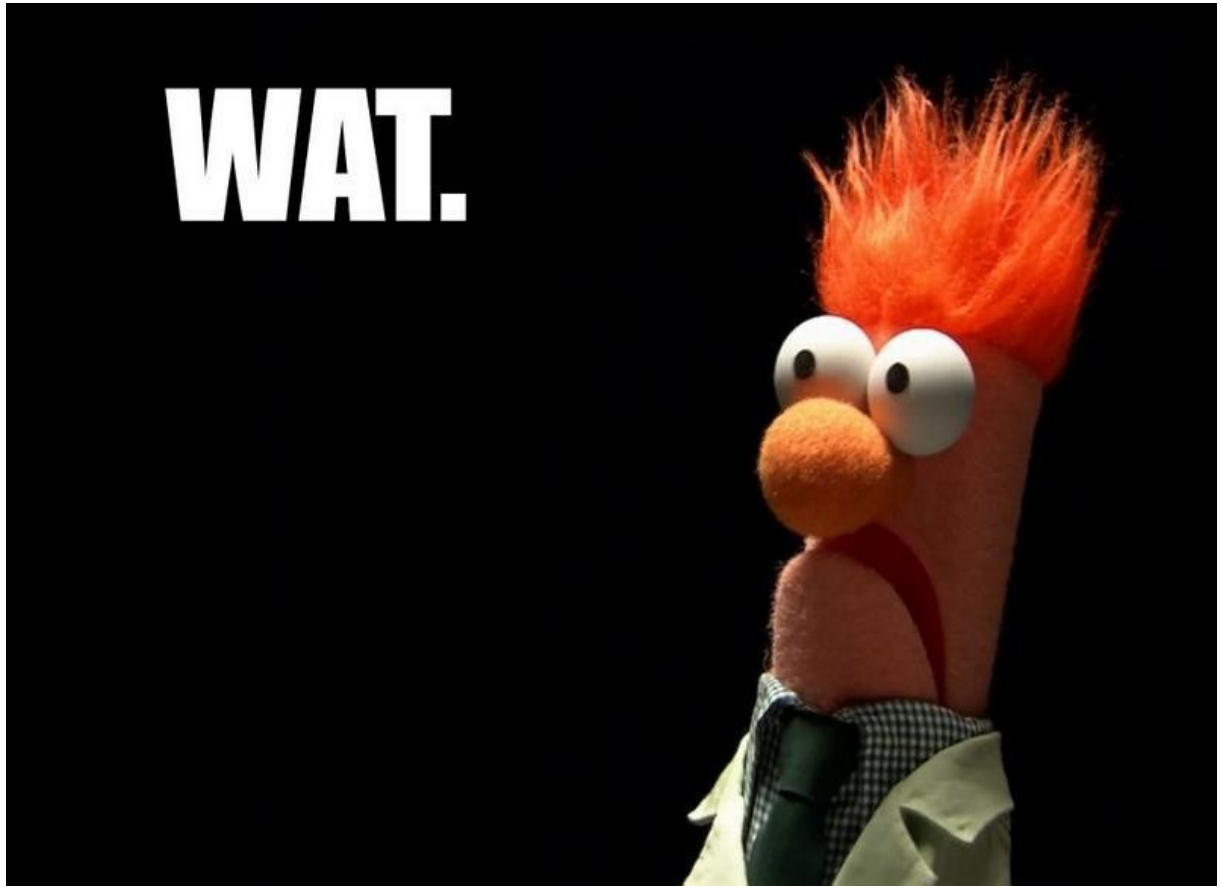
Un po' di storia

(Non fa mai male)

Le LISP machine degli anni '70, teorizzate da John McCarthy e costruite da Richard Greenblatt, avevano tastiere con il tasto **`META`** che permett

“LISP”? “McCarthy”? “Greenblatt”? Eh?

(Is this Operating Systems 101 or Computer History 101?)



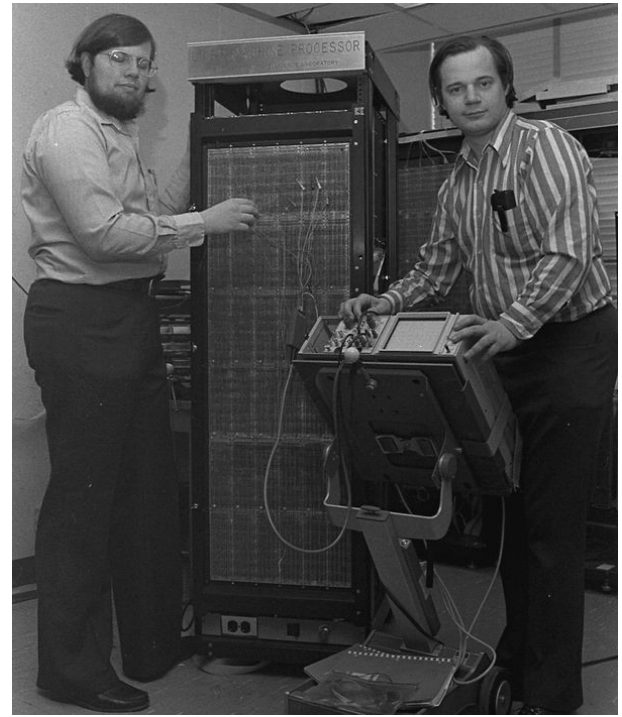
Un po' di storia

(Riparte il sermone da vecchio)

Negli anni '70, parallelamente ai mainframe si sono diffuse le **LISP machine**.

La prima LISP machine è stata **Knight**, progettata da Richard Greenblatt e prototipata da Thomas Knight.

La LISP machine esegue in maniera efficiente (supportata dall'hardware) programmi scritti nel linguaggio di programmazione **LISP**.



Greenblatt (a sinistra)
Knight (a destra)
CADR LISP machine (al centro)
MIT, 1978

Un po' di storia

(Riparte il sermone da vecchio)

LISP è stato inizialmente concepito da John McCarthy come uno strumento per scrivere pseudocodice.

Fortemente influenzato dal λ-calcolo di Alonzo Church.

La prima implementazione è dovuta a Steve Russell (quello di Spacewar!).

Alonzo Church
(1903-1995)

John McCarthy
(1927-2011)

Steve Russell
(1937-)

Un po' di storia

(Riparte il sermone da vecchio)

Cosa ha di così speciale LISP da indurre allo sviluppo di una architettura hardware fatta su misura?
È impossibile riassumere in una slide l'influenza di LISP in ambito informatico e di intelligenza artificiale.
Si rimanda gli interessati alla lettura della introduzione seguente:

Alan Kay
(1940-)

http://www.gigamonkeys.com/book/introduction-why-lisp.html

Alan Kay ha detto: **"LISP is the Maxwell's Equations of software!"**.

Un po' di storia

(Riparte il sermone da vecchio)

Le LISP machine sono equipaggiate di tastiere innovative, in grado di rappresentare i simboli matematici usati in LISP e di controllare il dispositivo di output (una telescrivente o un monitor primitivo).

Tastiera della Knight LISP machine (1974).

Un po' di storia

(Riparte il sermone da vecchio)

Si noti la presenza di un simbolo matematico sopra ogni lettera. Questi simboli sono usati a mo' di variabili e operatori nei programmi LISP.

Tastiera della Knight LISP machine (1974).

Un po' di storia

(Riparte il sermone da vecchio)

Il tasto **`CTRL`**, se premuto insieme ad un carattere, spegne i due bit più significativi del suo codice ASCII a 7 bit.

→ Vengono selezionati solo i caratteri con codice ASCII in [0, 31].

Tali caratteri (detti **di controllo**) controllano il terminale.

Tastiera della Knight LISP machine (1974).

Un po' di storia

(Riparte il sermone da vecchio)

Il tasto **`META`**, se premuto insieme ad un carattere, accende l'ottavo bit (altrimenti sempre spento).
→ Vengono selezionati solo i caratteri con codice ASCII in [128, 255].
Tali caratteri sono associati a comandi.

Tastiera della Knight LISP machine (1974).

Un po' di storia

(Riparte il sermone da vecchio)

Con l'avvento del Personal Computer IBM (più potente, più versatile, meno costoso), le LISP machine si avviano verso il loro inesorabile declino.

IBM PC 5150 (1981)

Un po' di storia

(Riparte il sermone da vecchio)

Si avvia verso l'inesorabile declino anche il tasto **`META`**, sostituito dal tasto **`Alt`**.

Tastiera IBM
Model F

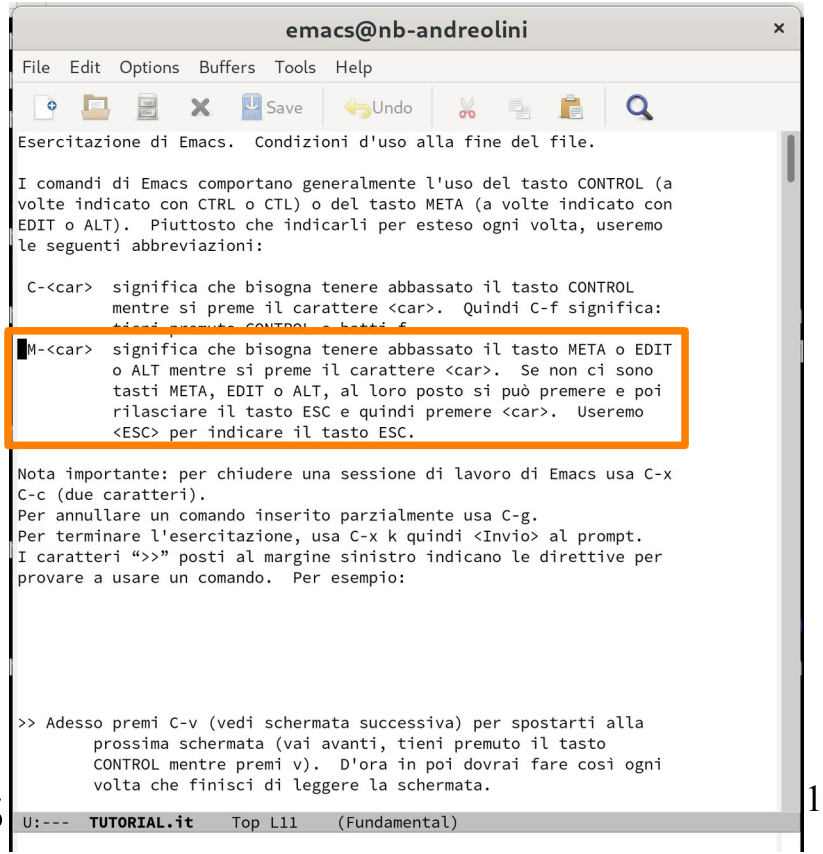
Un po' di storia

(Riparte il sermone da vecchio)

Il tasto **META** potrà anche essere scomparso, ma la sua leggenda rimane!

Diverse applicazioni mappano i comandi sulle sequenze **<Alt>-<Tasto>**.

Diversi manuali utente adottano ancora la terminologia **<META>-<Tasto>** o **M-<Tasto>**.

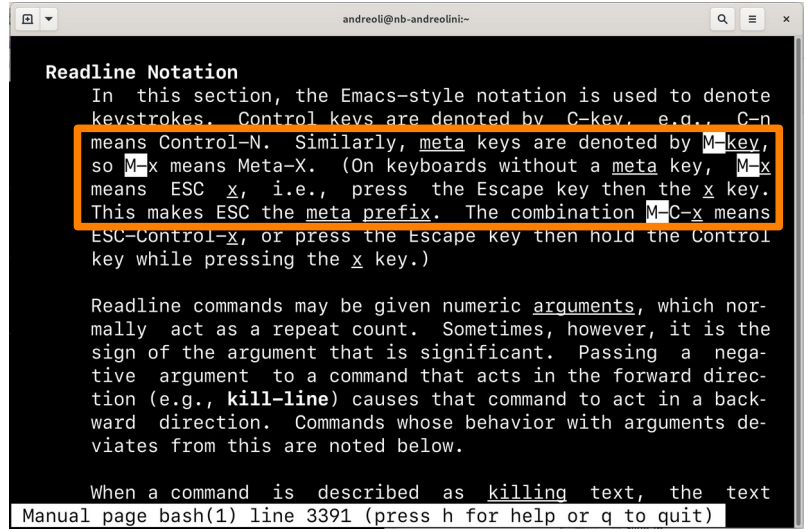


Tutorial di Emacs

Un po' di storia

(Riparte il sermone da vecchio)

Il tasto **META** potrà anche essere scomparso, ma la sua leggenda rimane!
Diverse applicazioni mappano i comandi sulle sequenze **<Alt>-<Tasto>**.
Diversi manuali utente adottano ancora la terminologia **<META>-<Tasto>** o **M-<Tasto>**.

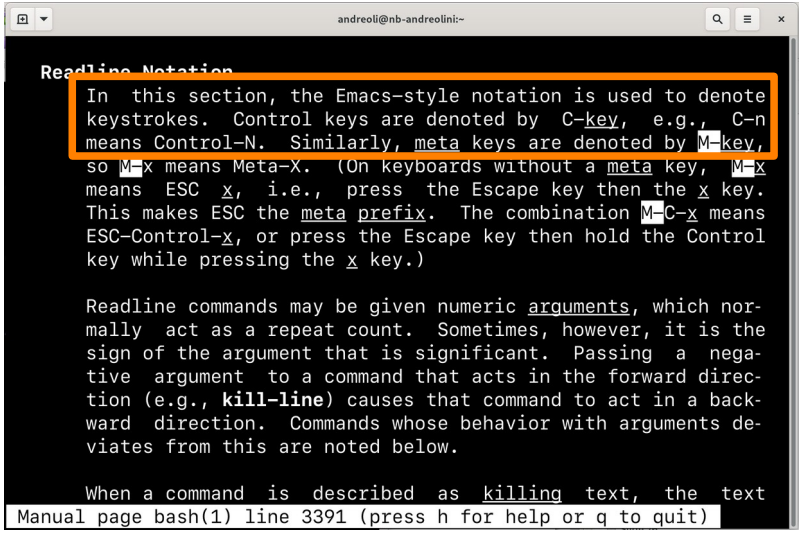


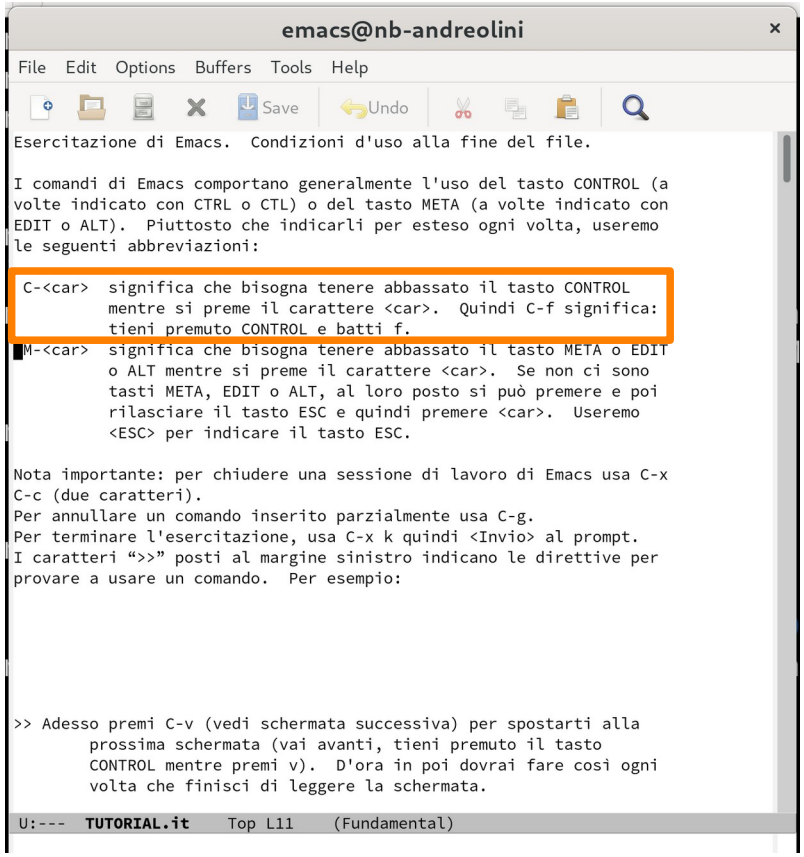
Pagina di manuale di BASH (funzionalità "readline")

Un po' di storia

(Riparte il sermone da vecchio)

Si usa associare comandi anche a **<CTRL>-<Tasto>** o **C-<Tasto>**.

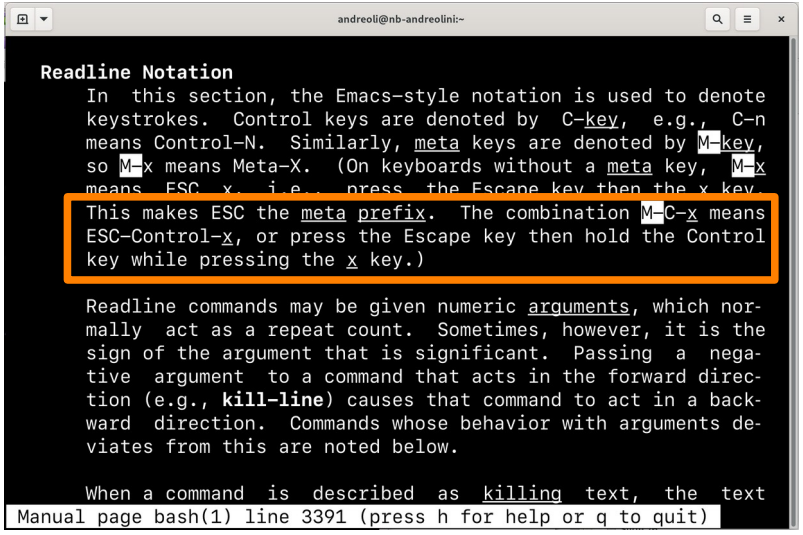




Un po' di storia

(Riparte il sermone da vecchio)

Si usa associare comandi anche a **<CTRL>-<Alt>-<Tasto>** o **C-M-<Tasto>**. Nel caso specifico di BASH, il tasto **ESC** può essere sostituito ad **Alt**, e si hanno combinazioni del tipo **M-C-<Tasto>** (ovvero **ESC**, seguito da **<CTRL>-<Tasto>**).



Stampa caratteri non stampabili

(Si usa l'opzione **`-v`** di **`cat`** per stampare i caratteri non stampabili)

Per i più curiosi tra voi, il docente ha prodotto una tabella di conversione tra codice ASCII e rappresentazione di **`cat -v`**. In questo modo, si scoprono i vari caratteri usati nelle rappresentazioni dei simboli.

File **`cat-v-table.txt`** negli approfondimenti alla presente lezione.

Un esempio concreto

(Svolto passo passo, per non spaventare troppo gli installatori di ubuntu)

Si crei un file di nome **`file.txt`** con il contenuto seguente:

```
Questo è un file di testo.
```

Si stampano i caratteri speciali di **`file.txt`**:

```
cat -v file.txt
```

In un SO moderno si dovrebbe ottenere l'output seguente:

```
Questo M-CM-( un  file di testo.
```

Un esempio concreto

(Svolto passo passo, per non spaventare troppo i configuratori di facebook)

L'output rivela dei caratteri speciali, identificati con la sequenza **`M-C`** e **`M-(`**:

```
Questo M-CM-( un  file di testo.
```

Perché una sequenza di due byte per un singolo carattere "è"?
Il carattere "è" non è nell'insieme ASCII standard, pertanto deve essere codificato. L'encoding di default nei SO moderni è UTF-8 (multi-byte, in questo caso due byte).

A quali byte corrispondono queste due codifiche?

Un esempio concreto

(Svolto passo passo, per non spaventare troppo i valutatori di app android)

Un metodo immediato di individuare i byte associati alla codifica consiste nell'uso della tabella **`cat-v-table.txt`**:

```
...
M-(         168             a8
...
M-C         195             c3
...
```

I due byte codificati sono 195 e 168.

Un esempio concreto

(Svolto passo passo, per non spaventare troppo gli utenti di cali linux)

Il metodo classico consiste nell'applicare l'operazione originaria associata al tasto **META**, ovvero accendere l'ottavo bit del carattere.
Si ha:

```
ASCII('(') | 128 = 40 | 128 = 168
ASCII('C') | 128 = 67 | 128 = 195
```

I due byte codificati sono 195 e 168.

La codifica UTF-8 di “è”

(La si può verificare con tabelle online)

La codifica UTF-8 del carattere “è” può essere verificata consultando una delle tante tabelle online, ad esempio:

http://www.fileformat.info/info/unicode/index.htm

Si clicca il link “Search”, si immette il carattere “è” e si preme il pulsante “Search”.
Si ottiene una corrispondenza Unicode avente codice U+00E8; la si clicca per ottenere le codifiche corrispondenti.
Si ottiene la codifica UTF-8 c3 a8 (195 168).

Una domanda pertinente

(E annessa risposta)

Se i caratteri di controllo si possono immettere da terminale con la sequenza **<CTRL>-<Tasto>**, i caratteri estesi si possono immettere da terminale con la sequenza **<Alt>-<Tasto>**?

Risposta breve: una volta sì, oggi no.
La codifica dei caratteri usata di default nei sistemi moderni è quella universale Unicode, che usa a sua volta un algoritmo di codifica multi-byte dal nome UTF-8.

Repetita iuvant.

Immissione caratteri tramite Unicode

(Si usa la sequenza **<CTRL>-<SHIFT>-u**, seguita dal codice Unicode)

Le applicazioni concordano nel mettere a disposizione una sequenza di tasti ben specifica per immettere un qualunque carattere Unicode a partire dal suo codice.
Si prema **<CTRL>-<SHIFT>-u** (dovrebbe comparire una lettera u sottolineata). Si immetta il codice ASCII o Unicode in esadecimale e **<INVIO>**.

Ad esempio, per la lettera "è":
<CTRL>-<SHIFT>-u e8<INVIO>

Stampa caratteri non stampabili

(Ancora?)

L'opzione **`-A`** di **`cat`** elenca tutti i caratteri non stampabili visti finora. Essa è equivalente a **`-vET`**.

Ad esempio, per vedere tutti i caratteri non stampabili del file **`file.txt`**, si esegue il comando seguente:

```
cat -A file.txt
```

Si ottiene l'output seguente:

```
Questo M-CM-( un file di testo.$
```

Stampa dei numeri di riga

(Si usa l'opzione **`-n`** di **`cat`**)

L'opzione **`-n`** di **`cat`** prepende un numero progressivo ad ogni riga del file.

Ad esempio, per numerare le righe del file **`/etc/passwd`** si esegue il comando:

```
cat -n /etc/passwd
```

Si ottiene l'output seguente:

```
1   root:x:0:0:root:/root:/bin/bash
2   daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
3   bin:x:2:2:bin:/bin:/usr/sbin/nologin
...
```

Esercizio 10 (2 min.)

Leggendo la pagina di manuale del comando opportuno, individuate un modo per stampare un file "al contrario" (dall'ultima alla prima riga).
Applicate tale metodo per stampare le righe del file **`/etc/passwd`** dall'ultima alla prima.

Visualizzazione di file binari

(Si usa il comando **`hexdump`**)

Il comando **`hexdump`** stampa il contenuto di file in diversi formati. Nella sua accezione più semplice, **`hexdump`** è lanciato senza opzioni e con un elenco di nomi di file esistenti **F_1**, **F_2**, … **F_N**. In tali condizioni, **`hexdump`** stampa il contenuto **F_1**, **F_2**, … **F_N** in esadecimale, a parole di 16 bit:

`hexdump` **F_1** **F_2** **`...`** **F_N**

Un esempio concreto

(Più che mai necessario, qui)

Ad esempio, per visualizzare in esadecimale il contenuto del file **`/etc/hostname`**, si esegue il comando:

```
hexdump /etc/hostname
```

Si ottiene l'output seguente:

```
0000000 6564 6962 6e61 000a
0000007
```

Offset in esadecimale

Byte in esadecimale ("debian\n")

Una osservazione sottilissima

(Da punto bonus)

Gli studenti più attenti si saranno accorti di un dettaglio molto particolare. I byte componenti le parole rappresentano a due a due le parole "Debian", rovesciate! Perché l'output è così?

Offset in esadecimale

In realtà è "ed""ib""na"

Una osservazione sottilissima

(Da punto bonus)

L'output di default di **`hexdump`** è a parole esadecimali di 16 bit. Poiché l'architettura Intel è Little Endian, i numeri sono rappresentati con i bit meno significativi per primi.
→ I due caratteri "de" sono presi, convertiti in due codici ASCII 0x64 e 0x65, rappresentati in Little Endian (0x65, 0x64) e stampati.
Idem per i caratteri "bi" e "an".

Visualizzazione canonica

(Si usa l'opzione **`-C`** di **`hexdump`**)

L'opzione **`-C`** di **`hexdump`** abilita la visualizzazione in forma canonica, ovvero la combinazione di output esadecimale e stringa dei singoli byte.
L'output ha la forma seguente:

```
Offset    Dump esadecimale |dump_stringa  |
00000000  <16 byte hex.>   |<16 caratteri>|
00000010  ...               …
```

Attenzione! Poiché qui il flusso è per singoli byte e non per parole da 16 bit, il problema della endianness non si pone e i codici ASCII sono già nella sequenza giusta.

Una osservazione esempio concreto

(Più che mai necessario, anche qui)

Ad esempio, per visualizzare in esadecimale il contenuto del file **`/etc/hostname`**, si esegue il comando:

```
hexdump -C /etc/hostname
```

Si ottiene l'output seguente:

```
0000000 6465 6269 616e 000a                     |debian.|
0000007
```

Offset in esadecimale

Byte in esadecimale ("debian\n") nella sequenza giusta

Caratteri corrispondenti

Visualizzazione verbosa

(Si usa l'opzione **-v** di **hexdump**)

Per motivi di efficienza e di compattezza, **hexdump** non stampa righe ripetute di output, ma le sostituisce con il carattere asterisco *****.

Ad esempio, si visualizzi **/bin/ls**:

```
hexdump /bin/ls
```

All'offset 250 c'è una compattazione, poiché il contenuto in [240, 24f] è identico a quello in [250, 25f]:

```
0000240 0000 0000 0000 0000 0000 0000 0000 0000
*
0000260 0000 0000 0000 0000 0010 0000 0000 0000
```

Visualizzazione verbosa

(Si usa l'opzione **`-v`** di **`hexdump`**)

L'opzione **`-v`** di **`hexdump`** abilita la visualizzazione verbosa, di fatto eliminando le compattazioni.

Con riferimento all' esempio precedente:

```
hexdump -v /bin/ls
```

All'offset 250 si ripete il contenuto dell'offset 240:

```
0000240 0000 0000 0000 0000 0000 0000 0000 0000
0000250 0000 0000 0000 0000 0000 0000 0000 0000
0000260 0000 0000 0000 0000 0010 0000 0000 0000
```

Esercizio 11 (3 min.)

Visualizzate nella forma canonica di **`hexdump`** i primi 32 byte dei file seguenti:

`/usr/bin/ls`
`/usr/bin/bash`
`/usr/bin/which`

Notate qualche differenza?

Visualizzazione parte iniziale

(Si usa il comando **head**)

Il comando **head** visualizza la parte iniziale di un file. Nella sua accezione più semplice, **head** è lanciato senza opzioni e con un elenco di nomi di file esistenti **F1**, **F2**, … **FN**. In tali condizioni, **head** stampa le prime dieci righe di **F1**, **F2**, … **FN**:

```
head F1 F2 ... FN
```

Ad esempio, per visualizzare le prime dieci righe del file **/etc/passwd**, si esegue il comando seguente:

```
head /etc/passwd
```

Impostazione del numero di righe

(Si usa l'opzione **`-n`** del comando **`head`**)

L'opzione **`-n`** di **`head`** riceve un argomento **`N`**, il numero di righe da stampare.

- **`N`** positivo: stampa le prime **`N`** righe.
- **`N`** negativo: stampa fino alle ultime **`N`** righe.

Ad esempio, per stampare solo la prima riga di **`/etc/passwd`**, si esegue il comando seguente:

```
head -n 1 /etc/passwd
```

Invece, per stampare dalla prima alla penultima riga:

```
head -n -1 /etc/passwd
```

Impostazione del numero di byte

(Si usa l'opzione **-c** del comando **head**)

L'opzione **-c** di **head** è concettualmente simile a **-n**. Essa riceve un argomento **C**, il numero di byte da stampare.

C positivo: stampa i primi **C** byte.

C negativo: stampa fino agli ultimi **C** byte.

Ad esempio, per stampare solo i primi dieci byte di **/etc/passwd**, si esegue il comando seguente:

```
head -c 10 /etc/passwd
```

Invece, per stampare dal primo al terzultimo byte:

```
head -c -3 /etc/passwd
```

Visualizzazione parte finale

(Si usa il comando **`tail`**, concettualmente duale a **`head`**)

Il comando **`tail`** visualizza la parte finale di un file; esso è concettualmente duale a **`head`**.

Siano **`N`** un numero di righe e **`C`** un numero di byte. Si ha:

- **`-n N`**: stampa le ultime **`N`** righe.
- **`-n +N`**: stampa a partire dalla riga **`N`**.
- **`-c C`**: stampa gli ultimi **`C`** byte.
- **`-c +C`**: stampa a partire dal byte **`C`**.

Visualizzazione dinamica parte finale

(Si usa l'opzione **`-F`** del comando **`tail`**)

Il comando **`tail`** ha un'ulteriore opzione (**`-F`**) per la visione di file di log. Tale opzione svolge le seguenti operazioni:

continua a provare l'apertura del file fino a quando non viene creato (se non esiste già);
mostra le ultime righe del file;
si blocca in attesa di nuove righe da mostrare.

Un esempio concreto

(Vale più di $e^{6,907755279}$ parole)

Si provi a monitorare il file **`log.txt`**, anche se non esiste (anzi, meglio!):

```
tail -F log.txt
```

Si apre un'istanza di Gedit, si edita il buffer con un contenuto arbitrario terminato da **`<INVIO>`** e lo si salva nel file **`log.txt`**.

L'output di **`tail`** dovrebbe annunciare la creazione del file e la stampa del buffer.

Ogni ulteriore riga viene evidenziata nell'output di **`tail`** non appena viene scritta.

Esercizio 12 (3 min.)

Individuate due modi distinti in **`head`** per stampare i primi 1024 byte del file **`/bin/ls`**.

MANIPOLAZIONE DI TESTI

Scenario e interrogativi

(Quali strumenti sono disponibili per manipolare un file testuale?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione per manipolare contenuti testuali.

Interrogativi:

Quali sono tali strumenti? Come funzionano? Su quale tipologia di file operano esattamente?

Archivi testuali

(Record testuali separati dal fine linea; singoli campi separati da un separatore)

Nei SO UNIX-like la configurazione di un sottosistema può presentarsi sotto forma di **archivio testuale**.
Archivio testuale: è un file di testo contenente **record**.
Record: è un elenco di **campi**, separati da un carattere di separazione. Il record termina con il fine linea.
Campo: è una informazione riguardanti un attore del sottosistema.

Esempi di archivi testuali:

- **`/etc/passwd`** (archivio degli utenti)
- **`/etc/group`** (archivio dei gruppi di lavoro)

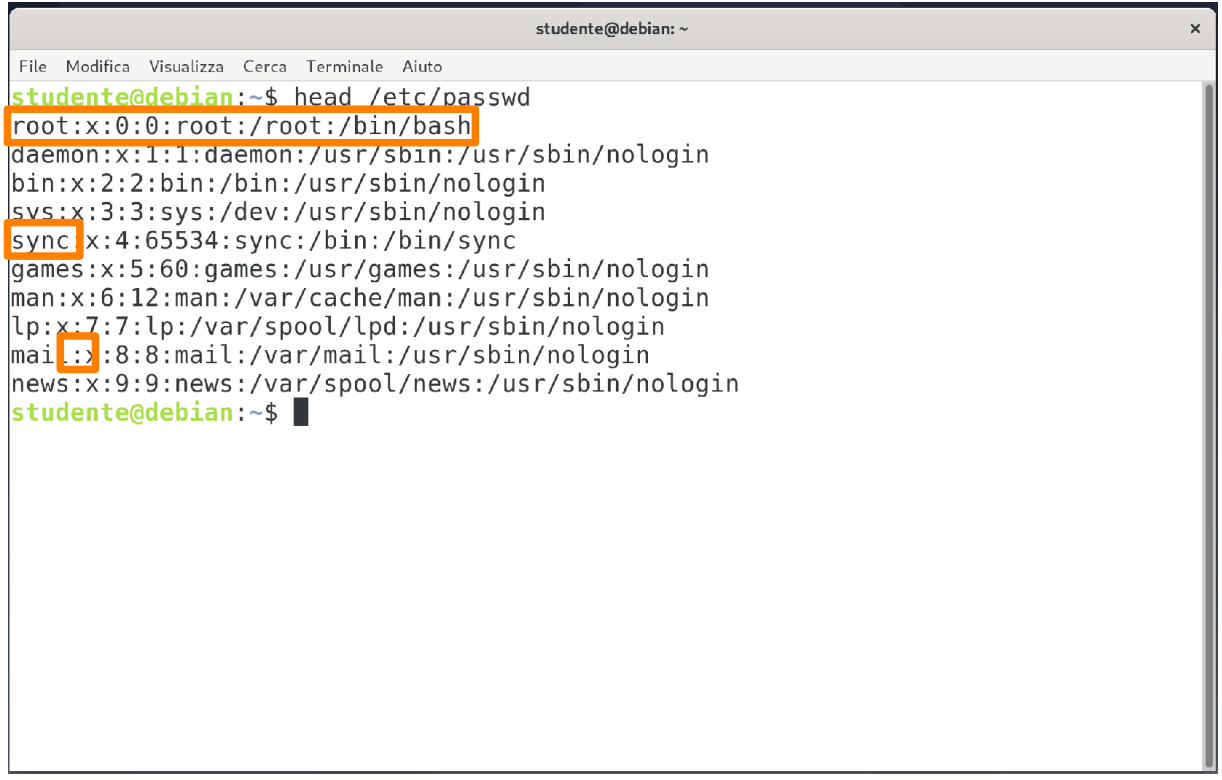
Un esempio concreto

(L'archivio testuale **`/etc/passwd`**)

Record

Campo

Carattere di separazione (**`:`**)



Selezione di elementi di un file

(Si usa il comando **`cut`**)

Il comando **`cut`** estrae elementi selezionati da uno o più file esistenti **`F1`**, **`F2`**, … **`FN`**:

`cut [opzioni] F1 F2 … FN`

Le opzioni servono per lo più ad identificare gli elementi da selezionare. Gli elementi possono essere identificati in modi diversi:

- intervalli di campi di un archivio testuale;
- intervalli di byte;
- intervalli di caratteri.

Una osservazione importante

(Qualche studente si sarà già posto la domanda)

Perché selezionare per intervalli di byte e di caratteri? Non è la stessa cosa?

No, non lo è. Le codifiche moderne dei caratteri sono multi-byte. Non è detto che un carattere corrisponda ad un singolo byte!

Selezione di campi da archivi testuali

(Si usano le opzioni **`-f`** e **`-d`** di **`cut`**)

L'opzione **`-f`** di **`cut`** permette di selezionare i campi da un file. L'argomento è una specifica sintetica di un intervallo di campi.

- **`-f N`**: l'N-mo campo
- **`-f N-`**: dall'N-mo campo a fine riga
- **`-f N-M`**: dall'N-mo campo all'M-mo campo
- **`-f N,M`**: l'N-mo campo e l'M-mo campo
- **`-f -M`**: dal primo campo all'M-mo campo

Il carattere di delimitazione di default è **`<TAB>`**; si può cambiare con l'opzione **`-d`**.

Un esempio concreto

(Laddove leggete **<TAB>**, premete il tasto **<TAB>**)

Si crei un file **db.txt** con il contenuto seguente:

```
f11<TAB>f21<TAB>f31
f12<TAB>f22<TAB>f32
```

Per selezionare il primo campo:

```
cut -f 1 db.txt
```

Per selezionare i primi due campi:

```
cut -f 1-2 db.txt
```

Per selezionare il primo ed il terzo campo:

```
cut -f 1,3 db.txt
```

Esercizio 13 (3 min.)

Stampate il primo ed il settimo campo di ogni riga dell'archivio testuale **`/etc/passwd`**.

Selezione di caratteri da file

(Si usa l'opzione `-c` di `cut`)

L'opzione **`-c`** di **`cut`** permette di specificare un intervallo di caratteri. L'argomento è una specifica sintetica dell'intervallo dei caratteri.

- **`-c N`**: N-mo carattere
- **`-c N-`**: dall'N-mo carattere a fine riga
- **`-c N-M`**: dall'N-mo carattere all'M-mo carattere
- **`-c N,M`**: l'N-mo carattere e l'M-mo carattere
- **`-c -M`**: dal primo carattere all'M-mo carattere

Esercizio 14 (1 min.)

Stampate i primi tre caratteri di ogni riga dell'archivio testuale **`/etc/passwd`**.

Selezione di byte da file

(Si usa l'opzione **`-b`** di **`cut`**)

L'opzione **`-b`** di **`cut`** permette di specificare un intervallo di byte. L'argomento è una specifica sintetica dell'intervallo dei byte.

- **`-b N`**: N-mo byte
- **`-b N-`**: dall'N-mo byte a fine riga
- **`-b N-M`**: dall'N-mo byte all'M-mo byte
- **`-b N,M`**: l'N-mo byte e l'M-mo byte
- **`-b -M`**: dal primo byte all'M-mo byte

Presenza di più separatori consecutivi

(Una trappola micidiale)

Si crei un file vuoto di nome **`db.txt`**. Si inseriscano le due righe seguenti nel file:

```
r1c1  r1c2  # due spazi separatori
r2c1   r2c2 # tre spazi separatori
```

Si salvi il file e si provi ad estrarre il secondo campo dell'archivio testuale:

```
cut -f2 -d" " db.txt
```

Il comando non ritorna alcun output.
Perché?

Una riflessione su **cut**

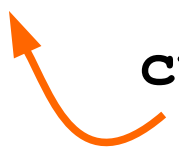
(Considera separatori vicini come un campo nullo)

Il comando **cut** si aspetta di trovare un unico separatore tra un campo ed il successivo.
Se trova più di un separatore contiguo, assume di avere un campo di valore nullo (NULL) in mezzo.

Due spazi → Spazio – valore nullo – spazio.

Di conseguenza, **cut** interpreta **db.txt** così:

```
r1c1 spazio NULL spazio r1c2
r2c1 spazio NULL spazio NULL spazio r2c2
```



```
cut -f2 -d" " db.txt
```

ritorna questa colonna

Costruzione di file multicolonnari

(Si usa il comando **paste**)

Il comando **paste** fonde uno o più file esistenti **F1**, **F2**, … **FN** in un unico output multicolonnare. Nella sua accezione più semplice, viene eseguito senza opzioni e con un elenco di file a singola colonna:

```
paste F1 F2 ... FN
```

Un esempio concreto

(Elenco numerato dei primi dieci utenti nel file **`/etc/passwd`**)

Si crei un file **`id.txt`** contenente la sequenza di numeri da 1 a 10 (un numero per riga).

Si crei un file **`users.txt`** contenente i primi dieci username in **`/etc/passwd`**.

Per creare un file multicolonnare a partire dalle due colonne in **`id.txt`** e **`users.txt`**, si esegue il comando:

```
paste id.txt users.txt
```

Esercizio 15 (2 min.)

Producete un output multicolonnare nel modo seguente.
Prima colonna. Una sequenza di numeri interi crescenti a partire da 1.
Seconda colonna. Gli username esistenti nel sistema (primo campo).
Terza colonna. Le shell assegnate agli username (ultimo campo).

Ordinamento di file per colonne

(Si usa il comando **`sort`**)

Il comando **`sort`** ordina file secondo specifici criteri. Nella sua accezione più semplice, **`sort`** viene lanciato senza opzioni e con uno o più file esistenti **`F1`**, **`F2`**, … **`FN`**. In tali condizioni, **`sort`** svolge le operazioni seguenti:

concatena i file **`F1`**, **`F2`**, … **`FN`**;
considera l'intera riga come una singola chiave di ordinamento;
ordina tutte le chiavi alfabeticamente.

Un esempio concreto

(Illustra l'ordinamento standard)

Si crei un file **`lista.txt`** con il contenuto seguente:

```
Frase1
Frase2
99
100
```

Per ordinare alfabeticamente il file si esegue il comando:

```
sort lista.txt
```

Criteri di ordinamento

(Too many to tell)

I criteri di ordinamento gestiti da **`sort`** sono innumerevoli. **`man sort`** per tutti i dettagli.

Si riportano i criteri di ordinamento più comuni:

- **`-n`**: ordinamento numerico
- **`-d`**: ordinamento alfanumerico (dizionario)
- **`-h`**: ordinamento numerico "umano" (confronta numeri leggibili dagli umani, ad es. 2G e 1K)
- **`-M`**: ordinamento del mese (confronta JAN, FEB, …)

Un esempio concreto

(Illustra l'ordinamento standard)

Si crei un file **`numeri.txt`** con il contenuto seguente:

```
7
3
99
112
```

Si ordini alfanumericamente il file:

```
sort numeri.txt
```

Si ordini numericamente il file:

```
sort -n numeri.txt
```

Si nota la differenza?

Ordinamento in base a chiave specifica

(Si usano le opzioni `-k` e `-t` di `sort`)

È possibile ordinare le righe di un file in base ad una chiave di ordinamento in esse contenuta.

- Campo di un archivio testuale.
- Insieme di caratteri in una specifica posizione.

L'opzione `-k` riceve come argomento una specifica testuale `KEY` del formato di una chiave, che illustra come recuperarla da una riga.

L'opzione `-t` specifica il carattere separatore dei campi (di default due campi sono separati da uno o più spazi).

Formato della chiave di ricerca

(Non esattamente immediato)

Il formato della chiave di ricerca ha una forma Backus-Naur del tipo **`POS1[,POS2]`**, ovvero **`POS1`** (obbligatorio) specifica la posizione iniziale e **`POS2`** (facoltativo) specifica la posizione finale.

`POS1` e **`POS2`** hanno una forma Backus-Naur del tipo **`F[.C][OPTS]`**, in cui:

`F` (obbligatorio) è il numero di campo desiderato;

`C` (facoltativo) è la posizione del carattere nel campo;

`OPTS` (facoltativo) è una lettera che corrisponde ad una opzione di ordinamento.

Un esempio concreto

(Ordinamento in base ad un ID numerico contenuto in un campo)

Si crei il file **`inventario.txt`** con il contenuto:

```
ID-23  descrizione1  costo-a
ID-99  descrizione2  costo-b
ID-112 descrizione3  costo-c
```

Si vuole ordinare **`inventario.txt`** usando come chiave di ordinamento il valore intero attaccato all'ID.

```
ID-23
ID-99
ID-112
```

Individuazione della chiave

(Alquanto complicata, la prima volta)

Di default **`sort`** separa i campi tramite lo spazio, pertanto non c'è bisogno di impostare un carattere separatore.

L'ID numerico richiesto è contenuto nel primo campo:

```
-k 1
```

L'ID numerico inizia al quarto carattere:

```
-k 1.4
```

L'ID numerico termina al sesto carattere:

```
-k 1.4,1.6
```

Si imposta l'ordinamento numerico per questa chiave:

```
-k 1.4,1.6n
```

Ordinamento tramite chiave

(Si usa il comando **`sort`** con la chiave appena impostata)

Per ordinare il file con la chiave specifica, si esegue il comando:

```
sort -k 1.4,1.6n inventario.txt
```

Si noti la differenza con il comando seguente, in cui viene provato un ordinamento numerico sul contenuto dell'intera riga:

```
sort -n inventario.txt
```

Ordinamento tramite chiave

(Un utilissimo addendum)

Se **`POS2`** non è specificata, si assume il fine riga. Ad esempio, se **`inventario.txt`** ha come contenuto:

```
descrizione1 costo-a ID-23
descrizione2 costo-b ID-99
descrizione3 costo-c ID-112
```

si può ordinare sulla parte numerica di ID così:

```
sort -k 3.4n -t " " inventario.txt
```

Esercizio 16 (2 min.)

Ordinate nel modo seguente il file **`/etc/passwd`**.
Ordinamento: numerico crescente.
Campo: quarto.
Separatore: due punti.

INDIVIDUAZIONE E RICERCA

Scenario e domande

(Quali strumenti sono disponibili per l'individuazione di file e directory?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione per individuare file e directory a partire dal nome e dai contenuti.

Interrogativi:

Quali sono tali strumenti? Come funzionano?
Gli strumenti per file e directory sono simili?

Individuazione di file dai metadati

(Si usa il comando **`find`**)

Il comando **`find`** individua file a partire dai suoi metadati. La sua sintassi è più complessa rispetto ai comandi tradizionali.

`find [opzioni] [percorso] [espressione]`

Opzioni: configurano le modalità operative di **`find`**.

Percorso: percorso iniziale di ricerca.

Espressione: criterio di individuazione su metadati.

Individuazione di tutti i file

(Nella sottodirectory attuale)

Il modo più semplice di eseguire **`find`** consiste nel lanciarlo senza opzioni e argomenti:

```
find
```

Tale comando assume che:

- la directory di ricerca sia quella attuale.
- il criterio di individuazione sia "elenca tutto".

Individuazione di tutti i file

(In una sottodirectory arbitraria)

Si provi ad aggiungere un primo argomento che imposta una sottodirectory arbitraria (ad esempio, **`/etc/`**):

```
find /etc
```

Tale comando:

imposta la directory di ricerca ad **`/etc`**.

Imposta il criterio di individuazione "elenca tutto".

Individuazione di alcuni file

(Aventi nomi particolari, in base a shell pattern)

Si provi ad aggiungere un ulteriore argomento che imposta un criterio di individuazione.
Il criterio più semplice è il match di un pattern di shell (**`-name` *`pattern`***):

```
find /etc -name *.conf
```

Tale comando:

imposta la directory di ricerca ad **`/etc`**.
Imposta il criterio di individuazione "elenca tutti i file che verificano il pattern *.conf".

Esercizio 17 (2 min.)

Nell'ipotesi che un file di log verifichi il pattern di shell `*.log`, individuate tutti i file di log nel sistema.

Individuazione di alcuni file

(Aventi nomi particolari, in base ad espressioni regolari)

È possibile individuare file anche in base al match di una espressione regolare, usando l'opzione **`-regex REGEX`**.
NOTA BENE: l'espressione regolare è applicata al nome di file individuato da **`find`**.

Se si esegue **`find .`** i file individuati hanno il formato seguente: **`./percorso`**.

Se si esegue **`find /etc`** i file individuati hanno il formato seguente: **`/etc/percorso`**.

Espressioni regolari supportate

(Sono tantissime)

Il comando **`find`** supporta una marea di formati di espressioni regolari. Il comando seguente li elenca tutti:

```
find -regextype help
```

L'output è il seguente:

```
... 'findutils-default', 'ed', 'emacs',
'gnu-awk', 'grep', 'posix-awk', 'awk',
'posix-basic', 'posix-egrep', 'egrep',
'posix-extended', 'posix-minimal-basic',
'sed'
```

Espressioni regolari supportate

(Sono tantissime)

Il formato di default usato da **`find`** è **emacs**:

```
... 'findutils-default', 'ed', 'emacs',
'gnu-awk', 'grep', 'posix-awk', 'awk',
'posix-basic', 'posix-egrep', 'egrep',
'posix-extended', 'posix-minimal-basic',
'sed'
```

Espressioni regolari Emacs

(Come studiarle, come provarle)

Gli studenti interessati possono riferirsi alla documentazione ufficiale GNU per conoscere il formato esatto delle diverse famiglie di espressione regolari:

https://www.gnu.org/software/findutils/manual/html_mono/find.html#Regular-Expressions

Gli studenti interessati possono provare le espressioni regolari nel modo seguente.

Si esegue il comando **`emacs`**.

Si clicca sul link "Emacs Tutorial".

Si attiva la ricerca per regexp in avanti con la combinazione di tasti **`<CTRL>-<Alt>-S`**.

Per interrompere la ricerca, si prema **`<CTRL>-g`**.

Espressioni regolari Emacs

(Quelle di base)

Le espressioni regolari di base nel formato Emacs accettano i seguenti caratteri speciali:

`.`	→	un carattere qualunque
`+`	→	il carattere o l'espressione regolare precedente, ripetuta almeno una volta
`?`	→	il carattere o l'espressione regolare precedente, ripetuta zero o una volta
`\+`	→	il carattere letterale `+`
`\?`	→	il carattere letterale `?`

Espressioni regolari Emacs

(Quelle di base)

Le espressioni regolari di base nel formato Emacs accettano i seguenti caratteri speciali:

- **`[c1-c2]`** → un carattere nell'intervallo [**`c1`**, **`c2`**]
- **`^`** → inizio riga
- **`$`** → fine riga
- **`R1\|R2`** → espressione regolare **`R1`** o **`R2`**
- **`\(REGEX\)`** → **`REGEX`** viene associata ad un blocco e può essere riferita in seguito
- **`\N`** → riferimento blocco N (N=1, 2, 3, …)

Espressioni regolari Emacs

(Estensioni GNU)

Le estensioni GNU alle espressioni regolari nel formato Emacs accettano i seguenti caratteri speciali:

- **`\w`** → un carattere dentro una parola
- **`\W`** → un carattere non dentro una parola
- **`\<`** → inizio di una parola
- **`\>`** → fine di una parola
- **`\b`** → bordo (inizio o fine parola)
- **`\B`** → interno (non inizio, non fine parola)

Scrittura dell'espressione regolare

(Non esattamente banale)

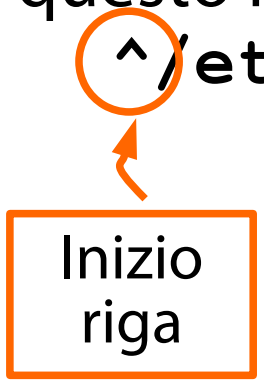
Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`



Scrittura dell'espressione regolare

(Non esattamente banale)

Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`

Scrittura dell'espressione regolare

(Non esattamente banale)

Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`

Scrittura dell'espressione regolare

(Non esattamente banale)

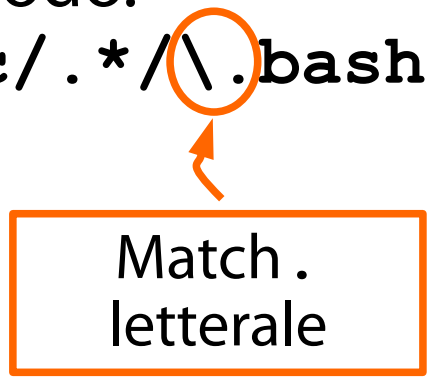
Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`



Scrittura dell'espressione regolare

(Non esattamente banale)

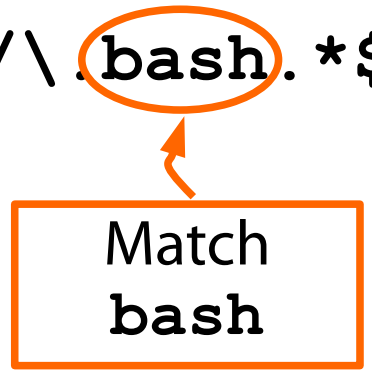
Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`



Scrittura dell'espressione regolare

(Non esattamente banale)

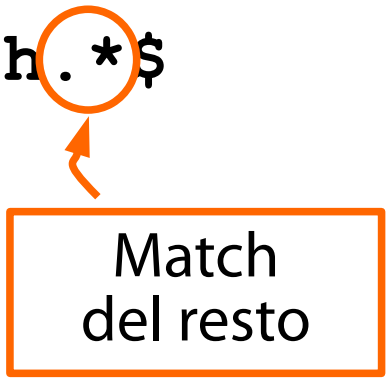
Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`



Scrittura dell'espressione regolare

(Non esattamente banale)

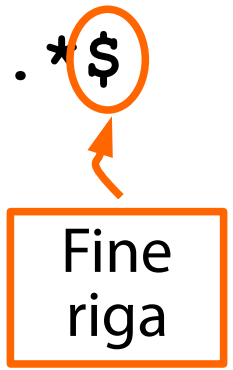
Ad esempio, si cerchino nella directory **`/etc`** tutti i file che iniziano con la stringa **`.bash`**.

Il formato dei file individuati da **`find`** è:

`/etc/directory/directory/.../.bash...`

L'espressione regolare deve catturare file espressi in questo modo.

`^/etc/.*/\.bash.*$`



Scrittura del comando

(Finalmente!)

In definitiva, il comando richiesto è:

```
find /etc -regex '^/etc/.*/\.bash.*$'
```

Individuazioni case-insensitive

(Espressioni -iname, -iregex)

Le espressioni **`-iname`** e **`-iregex`** sono le varianti "case insensitive" di **`-name`** e **`-regex`**.

Ad esempio, si provi a cercare il pattern di shell **`upower`** nella directory **`/etc`**:

`find /etc -name upower`

Non si trova nulla.

Si ritenti la ricerca in modalità "case insensitive":

`find /etc -iname upower`

Si trova il file **`/etc/UPower`**.

Esercizio18 (3 min.)

Individuate tutti i file HTML nel sistema.
In prima approssimazione, considerate come file HTML un file terminante in:

- `.html`
- `.htm`
- `.HTML`
- `.HTM`

Individuazione di alcuni file

(In base ad altri metadati)

È possibile individuare file anche in base a valori opportuni di altri metadati. Seguono alcuni esempi.

`-atime n`: il file è stato acceduto l'ultima volta n*24 ore fa.

`-perm mode`: il file ha permessi *mode*.

`-size n[cwbkMG]`: il file usa n unità di spazio su disco.

Semantica dei valori di n:

`+n`:	più grande di **`n`**
`-n`:	più piccolo di **`n`**
`n`:	esattamente pari a **`n`**

Esercizio 19 (2 min.)

Individuate tutti i file più grandi di 1MB presenti nell'intero sistema.

Specifica di azioni

(Sui file individuati)

Le espressioni possono esprimere sia criteri di individuazione, sia azioni da svolgere sui file individuati.

L'azione di default è **`-print`** (stampa i percorsi dei file).

Di azioni possibili ne esistono diverse: nel seguito sono elencate le più importanti.

Azione: stampa di metadati arbitrari

(In maniera analoga a quanto visto con il comando stat)

L'azione **`-printf`** accetta una stringa di formato che permette di personalizzare la stampa delle informazioni relative ai file individuati.

`man find` per tutti i dettagli.

Ad esempio, per stampare il nome del file ed i suoi permessi in ottale si digiti il comando seguente:

```
find / -printf "%p %m\n"
```

Esercizio 20 (3 min.)

Producete un elenco di tutti i file nel sistema con relativa dimensione:

```
/file1 dimensione1
/file2 dimensione2
...
```

Azione: esecuzione di un comando

(Un'azione potentissima)

L'azione **`-exec`** accetta un comando di shell da eseguire su ciascun file individuato.
Il comando contiene, al posto degli argomenti, la stringa **`'{}'`** (sostituita, di volta in volta, con il file individuato).
Il comando è terminato con la stringa **`\;`**.

```
find DIR EXPR -exec COMMAND '{}' \;
```

Volendo, si può anche evitare il quoting forte delle parentesi graffe **`{}`**. Poiché sono vuote, BASH non effettua alcuna espansione.

Un esempio concreto

(Stampa del tipo di ciascun file individuato)

Ad esempio, si supponga di voler identificare tutti i file di configurazione all'interno della directory **`/etc`**.
Per ciascun file individuato, se ne stampi il tipo (usando il comando **`file`**).
Si digiti il seguente comando (tutto su una riga):

```
find /etc -name "*.conf" -exec file "{}" \;
```

Esercizio 21 (2 min.)

Individuate tutti i file che verificano una delle proprietà seguenti:

- terminano con **`.bak`**;
- terminano con **`~`**.

Cancellate forzatamente i file individuati.

Individuazione di file dal contenuto

(Comando grep)

Il comando **grep** individua file a partire dal suo contenuto. La sua sintassi è più complessa rispetto ai comandi tradizionali.

grep [opzioni] pattern [file]

Opzioni: configurano le modalità operative di **grep**.

Pattern: espressione di ricerca.

File: uno o più file su cui operare.

Ricerca di una stringa in un file

(Uso basilare di grep)

Il modo più semplice di eseguire **grep** consiste nel lanciarlo senza opzioni, con un pattern semplice (una stringa) ed un file:

```
grep root /etc/passwd
```

Tale comando stampa tutte le righe del file **/etc/passwd** in cui è contenuta la stringa **root**.

Stampa della riga contenente il match

(Si usa l'opzione **`-n`** di **`grep`**)

L'opzione **`-n`** di **`grep`** stampa il numero di riga in cui è avvenuto un match con il pattern.

```
grep -n root /etc/passwd
```

L'output è presentato sotto forma di "database testuale" (un record con diversi campi, separati dal carattere `:`).

Stampa del file contenente il match

(Si usa l'opzione **`-H`** di **`grep`**)

L'opzione **`-H`** di **`grep`** stampa il nome del file in cui è avvenuto un match con il pattern.

```
grep -H root /etc/passwd
```

Tale opzione è molto utile quando si specifica una ricerca su una moltitudine di file:

```
grep -H root /etc/passwd /etc/group
```

Quando i file da ricercare sono più di uno, **`grep`** abilita di default l'opzione **`-H`** (che può pertanto essere omessa).

Ricerca ricorsiva in un sottoalbero

(Si usa l'opzione **`-R`** di **`grep`**)

L'opzione **`-R`** di **`grep`** effettua una ricerca ricorsiva nel sottoalbero specificato come argomento.

```
grep -R root /etc
```

Ricerca ricorsiva in un sottoalbero

(Si usa l'opzione **`-i`** di **`grep`**)

L'opzione **`-i`** di **`grep`** effettua una ricerca case-insensitive.

```
grep -i Root /etc/passwd
```

Colorazione dei risultati

(Si usa l'opzione **`--color=yes`** di **`grep`**)

L'opzione **`--color=yes`** di **`grep`** evidenzia i match in colore rosso.

```
grep --color=yes root /etc/passwd
```

Tale opzione è spesso utile per verificare la correttezza e l'efficacia dei match tramite espressione regolare.

Ricerca di una espressione regolare

(Si usa l'opzione **`-E`** di **`grep`**)

Il comando **`grep`** supporta tre diverse famiglie di espressioni regolari:

- base;
- estese;
- compatibili con il Perl.

In questo corso si discutono le espressioni regolari estese, introdotte dall'opzione **`-E 'REGEX'`**.

Espressioni regolari estese in **grep**

(Selezione di ripetizioni)

Le espressioni regolari estese di **grep** accettano i seguenti caratteri speciali:

- `.` → un carattere qualunque
- `+` → il carattere o l'espressione regolare precedente, ripetuta almeno una volta
- `?` → il carattere o l'espressione regolare precedente, ripetuta zero o una volta
- `*` → il carattere o l'espressione regolare precedente, ripetuta un qualunque numero di volte
- `\+` → il carattere letterale **+**
- `\?` → il carattere letterale **?**

Espressioni regolari estese in **grep**

(Selezione di ripetizioni)

Le espressioni regolari estese di **grep** accettano i seguenti caratteri speciali:

- **{N}** → il carattere o l'espressione regolare precedente, ripetuta esattamente N volte
- **{N,}** → il carattere o l'espressione regolare precedente, ripetuta almeno N volte
- **{M,N}** → il carattere o l'espressione regolare precedente, ripetuta da M a N volte

Espressioni regolari estese in **grep**

(Classi di caratteri)

Le espressioni regolari estese di **grep** accettano i seguenti caratteri speciali:

[c1-c2]	→	un carattere nell'intervallo [**c1**, **c2**]
[[:alnum:]]	→	un carattere alfanumerico
[[:alpha:]]	→	un carattere alfabetico
[[:blank:]]	→	un carattere "blank" (spazio, tabulazione)
[[:digit:]]	→	un carattere cifra
[[:lower:]]	→	un carattere alfabetico minuscolo
[[:space:]]	→	un carattere "space" (tab, newline, form feed, vertical tab, carriage return, spazio)
[[:upper:]]	→	un carattere alfabetico maiuscolo

Espressioni regolari estese in **grep**

(Ancore)

Le espressioni regolari estese di **grep** accettano i seguenti caratteri speciali:

- **^** → inizio riga
- **$** → fine riga
- **\w** → un carattere dentro una parola
- **\W** → un carattere non dentro una parola
- **\<** → inizio di una parola
- **\>** → fine di una parola
- **\b** → bordo (inizio o fine parola)
- **\B** → interno (non inizio, non fine parola)

Espressioni regolari estese in **grep**

(Blocchi)

Le espressioni regolari estese di **grep** accettano i seguenti caratteri speciali:

R1 | R2	→	espressione regolare **R1** o **R2**
(REGEX)	→	**REGEX** viene associata ad un blocco e può essere riferita in seguito
\N	→	riferimento blocco N (N=1, 2, 3, …)

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

Si chiede di trovare ed evidenziare in rosso tutti gli indirizzi IPv4 contenuti nel file **`/etc/hosts`**.
Si chiede inoltre di stampare il numero di riga per ogni match.

Che fare?

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

Per stampare il numero di riga del file si usa l'opzione **`-n`**.
Per evidenziare i match in rosso si usa l'opzione **`--color=yes`**.
Gli indirizzi IPv4 sono stringhe complesse, che devono essere intercettate tramite espressioni regolari.
Per descrivere la ricerca tramite una espressione regolare estesa, si usa l'opzione **`-E "REGEX"`** (dove **`REGEX`** è una opportuna espressione regolare).
Il template del comando da dare è il seguente:

```
grep --color=yes -nE "REGEX" /etc/hosts
```

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

Formato di un indirizzo IP:

`N1.N2.N3.N4`

dove **`Ni`** $\in$ [0, 255].

Scrivere una espressione regolare che intercetta i numeri da 0 a 255 è molto complicato.
Per semplificare l'esercizio, si scrive una espressione regolare che intercetta **`Ni`** $\in$ [0, 999].

```
([[:digit:]]{,3}.){3}[[:digit:]]{,3}
```

What the hell is this?

(Regexp magic!)

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

```
([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}
```

Intercetta un carattere cifra.

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

```
([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}
```

Intercetta fino a tre caratteri cifra consecutivi.

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

```
([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}
```

Intercetta fino a tre caratteri cifra consecutivi seguiti dal carattere "punto".

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

```
([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}
```

Considera il match precedente come un blocco.

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

```
([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}
```

Il blocco precedente (numero seguito da "**.**")
deve ripetersi tre volte.

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

```
([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}
```

Segue un carattere cifra ripetuto tre volte.

Un esempio concreto

(Si usano diverse opzioni di **`grep`** viste finora)

In definitiva, il comando richiesto è il seguente (da scrivere su una riga intera):

```
grep --color=yes -nE "([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}" /etc/hosts
```

Stampa esclusiva del match

(Opzione **-o** di **grep**)

L'opzione **-o** di **grep** stampa esclusivamente la porzione di riga che verifica il match.

```
grep -o root /etc/passwd
```

Questo comando stampa due righe contenenti il solo match **root**.

L'opzione **-o** è spesso usata per estrarre porzioni di testo interessanti, scartando le rimanenze.

Inversione del match

(Opzione **`-v`** di **`grep`**)

L'opzione **`-v`** di **`grep`** inverte il match impostato da linea di comando.

```
grep -v root /etc/passwd
```

Questo comando stampa tutte le righe che NON contengono la stringa **`root`**.

Esercizio 22 (4 min.)

Stampate i valori di tutte le etichette **`UUID`** nel file **`/etc/fstab`**.
Se possibile, evitate di stampare la stringa **`UUID=`**.

COLLEGAMENTI

Scenario e interrogativi

(Quali strumenti sono disponibili per il collegamento a file e a directory?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione per individuare collegamenti a file e a directory.

Interrogativi:

Quali sono tali strumenti? Come funzionano?

Definizione
(Collegamento a file)

I sistemi UNIX moderni consentono di creare **collegamenti a file**.
Collegamento (link): è una sorta di "puntatore" ad un file, tipicamente con un nome alternativo.

Usi tipici di un collegamento:

- fornire un percorso di file più semplice da scrivere
  `INTRO.gz` → `/usr/share/doc/bash/INTRO.gz`
- puntare ad uno tra N file alternativi
  `/usr/bin/X` → `/usr/bin/Xorg`

Tipologie di collegamento

(Collegamento fisico, collegamento simbolico)

Collegamento fisico (hard link). Il collegamento fisico è un nuovo elemento di directory che punta allo stesso contenuto del file originale. I due elementi di directory puntano allo stesso insieme di blocchi dati.

Collegamento simbolico (soft link). Il collegamento simbolico è un nuovo file, diverso da quello originale. Il contenuto del file è il percorso del file originale.

Collegamento fisico

(Si usa il comando **`ln`** senza l'opzione **`-s`**)

Il comando **`ln`** lanciato senza l'opzione **`-s`** crea collegamenti fisici:

`ln TARGET LINK_NAME`

dove:

`TARGET` è il percorso del file originale;

`LINK_NAME` è il nome del collegamento fisico.

Un esempio concreto

(Creazione di un collegamento fisico al file **`file.txt`**)

Ad esempio, si crei un file di nome **`file.txt`** con il contenuto seguente:

```
Questo è un file di testo.
```

Successivamente, si crei un collegamento di nome **`link_hard.txt`** a **`file.txt`**:

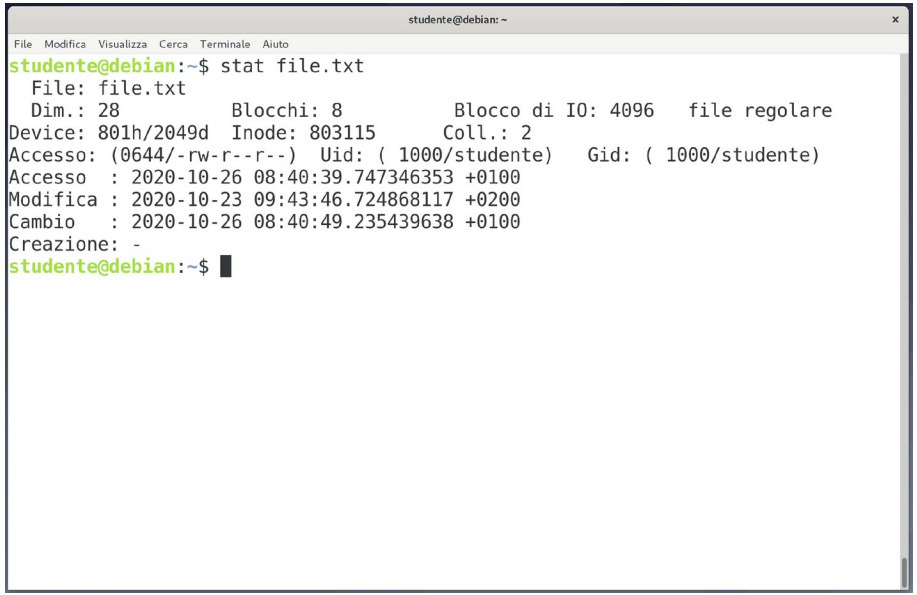
```
ln file.txt link_hard.txt
```

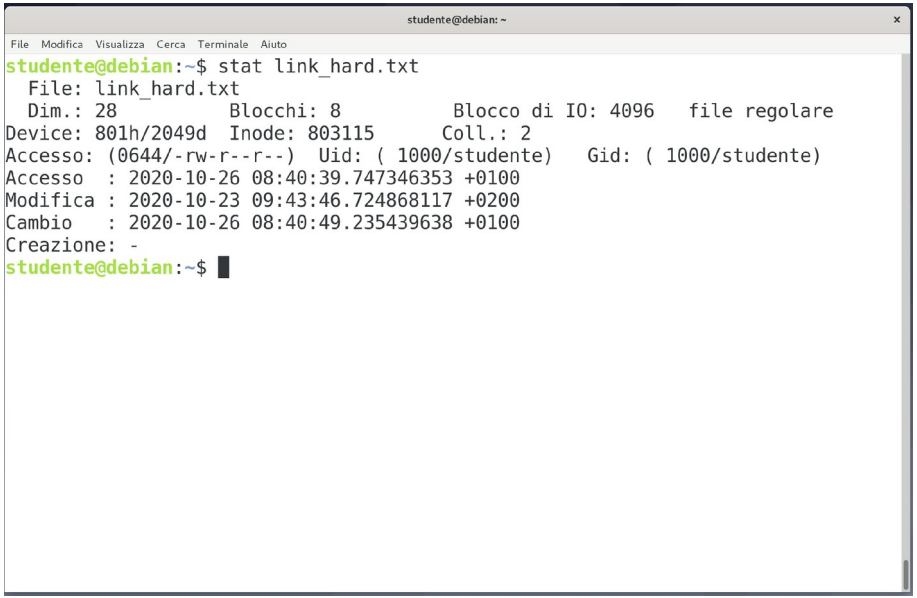
Indistinguibilità dei collegamenti fisici

(I metadati dei due file sono identici)

Si osservino i metadati dei due file appena creati:

`stat file.txt link_hard.txt`





I due file sono indistinguibili l'uno dall'altro!

Limitazioni dei collegamenti fisici

(Sono diverse)

Non si può creare un collegamento fisico ad una directory.
Non si può creare un collegamento fisico ad un file in un altro file system.
Non si può creare un collegamento fisico ad un file posseduto da un altro utente.

Tutte queste limitazioni sono superate dai collegamenti simbolici.

Collegamento simbolico

(Si usa l'opzione **`-s`** del comando **`ln`**)

L'opzione **`-s`** del comando **`ln`** permette la creazione di collegamenti simbolici:

```
ln -s TARGET LINK_NAME
```

dove:

`TARGET` è il percorso del file originale;

`LINK_NAME` è il nome del collegamento simbolico.

Un esempio concreto

(Creazione di un collegamento simbolico al file **`file.txt`**)

Si crei un collegamento di nome **`link_soft.txt`** a **`file.txt`**:

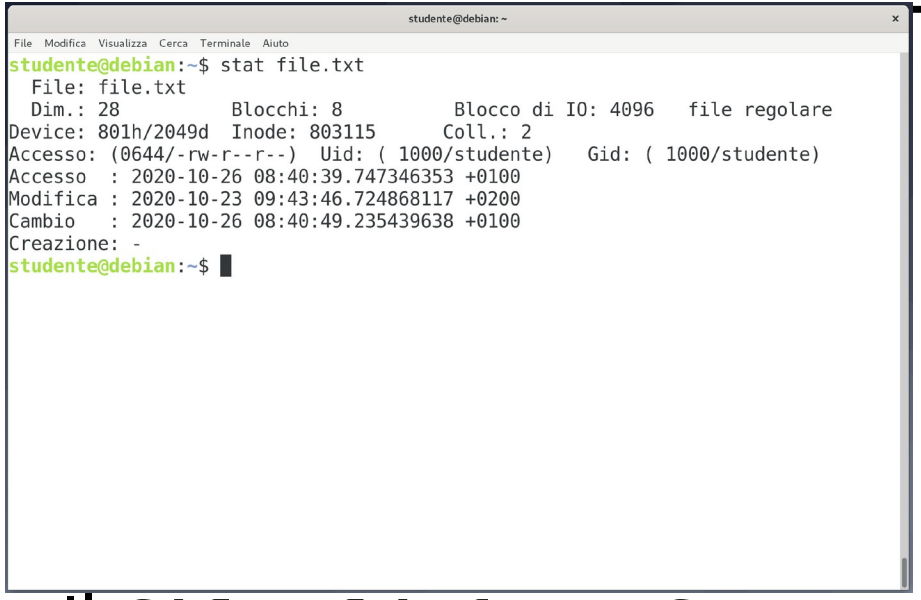
```
ln -s file.txt link_soft.txt
```

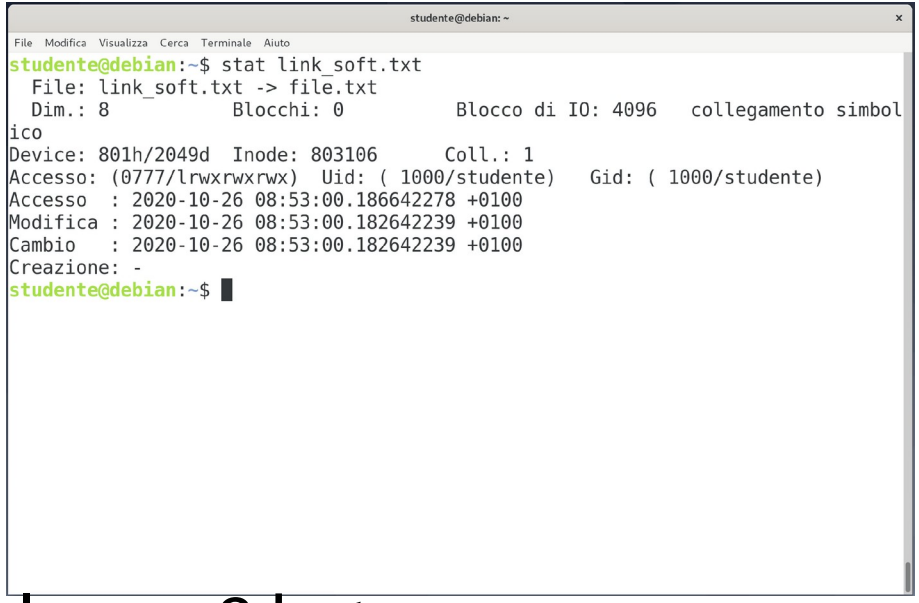
Differenza dei collegamenti simbolici

(I metadati dei due file sono diversi)

Si osservino i metadati dei due file appena creati:

`stat file.txt link_soft.txt`





Il **`file link_soft.txt`** è lungo 8 byte

→ lunghezza del percorso **`file.txt`**

Esercizio 23 (2 min.)

Create un collegamento fisico di nome **`passwd`** al file **`/etc/passwd`**. Ci riuscite?
Create un collegamento simbolico di nome **`passwd`** al file **`/etc/passwd`**. Ci riuscite?

Risoluzione collegamenti simbolici

(Si usa il comando **`readlink`**)

Il comando **`readlink`** risolve collegamenti simbolici. Nella sua accezione più semplice, **`readlink`** è lanciato senza opzioni e con un elenco di nomi di file esistenti **F_1**, **F_2**, ... **F_N**:

```
readlink F1 F2 ... FN
```

In tali condizioni, se **F_i** è un collegamento simbolico **`readlink`** stampa il file da esso puntato; altrimenti, non viene stampato nulla.

Un esempio concreto

(Stampa del file puntato da **`link_soft.txt`**)

Ad esempio, per risolvere il collegamento simbolico **`link_soft.txt`** si esegue il comando seguente:

```
readlink link_soft.txt
```

Si dovrebbe ottenere l'output seguente:

```
file.txt
```

Se si esegue **`readlink`** su un file regolare, non si ottiene alcun tipo di output:

```
readlink /etc/passwd
```

Risoluzione ricorsiva

(Si usa l'opzione **`-f`** di **`readlink`**)

Può capitare che un collegamento punti ad un altro collegamento simbolico, e così via.
Si consideri a puro titolo di esempio il collegamento simbolico **`/usr/bin/editor`**, che nei SO basati su Debian GNU/Linux punta all'editor di default di sistema.

```
readlink /usr/bin/editor
/etc/alternatives/editor
readlink /etc/alternatives/editor
/bin/nano
```

Bisogna risolvere ogni collegamento a mano? O si può automatizzare la procedura?

Risoluzione ricorsiva

(Si usa l'opzione **`-f`** di **`readlink`**)

L'opzione **`-f`** di **`readlink`** consente di risolvere i collegamenti in maniera ricorsiva:

```
readlink -f /usr/bin/editor
```

Si ottiene direttamente il file puntato dalla catena di collegamenti simbolici:

```
/bin/nano
```

ARCHIVI

Definizione

(Archivio)

I SO UNIX-like (tra cui GNU/Linux) mettono a disposizione diversi strumenti per la gestione di **archivi**.

Archivio: è un file che raccoglie uno o più file e/o directory nelle modalità seguenti.

- Memorizza i file e le directory nella gerarchia richiesta.
- Memorizza i metadati originali di file e directory.
- Memorizza informazioni suppletive per rilevare e correggere errori.
- È compresso opzionalmente per occupare meno spazio.

Usi di un archivio

(Sono diversi e tutti importanti)

Gli usi di un archivio sono molteplici:

trasferimento di una gerarchia di file e directory su un altro calcolatore;
invio di allegati complessi via posta elettronica;
copia di backup di file e directory importanti;
distribuzione pacchettizzata del software.

Il comando `tar`

(Gestisce archivi in formato TAR)

Un primo strumento UNIX per la gestione di archivi è il comando **`tar`**.

Il comando **`tar`** manipola archivi in formato TAR (detti anche **tarball**), opzionalmente compressi.

- Creazione.
- Estrazione.
- Elenco.

Specifica dell'archivio

(Si usa l'opzione **`-f`** di **`tar`**)

Ci si riferisce ad un archivio tramite l'opzione **`-f`** di **`tar`**, che riceve un argomento **`NAME`** contenente il percorso del file archivio (solitamente, un file di estensione **`.tar`**):

```
tar -f NAME
```

Ad esempio, per riferirsi ad un archivio di nome **`etc.tar`**, si scrive:

```
tar -f etc.tar
```

ATTENZIONE! Non eseguite questo comando; è incompleto!

Creazione di un archivio

(Si usa l'opzione **-c** di **tar**)

Per creare un archivio si usa l'opzione **-c** di **tar**, che accetta un elenco di file e/o directory **FD1**, **FD2**, … **FDN**.

```
tar -c -f NAME FD1 FD2 ... FDN
```

Ad esempio, per archiviare il contenuto della directory **/etc** (nei limiti consentiti dalle restrizioni di accesso ai file) nell'archivio **etc.tar**, si esegue il comando:

```
tar -c -f etc.tar /etc
```

Visione elenco file di un archivio

(Si usa l'opzione **`-t`** di **`tar`**)

Per vedere l'elenco dei file e directory in un archivio si usa l'opzione **`-t`** di **`tar`**:

```
tar -t -f NAME
```

Ad esempio, per elencare il contenuto dell'archivio **`etc.tar`** appena creato, si esegue il comando:

```
tar -t -f etc.tar
```

Estrazione di un archivio

(Si usa l'opzione **`-x`** di **`tar`**)

Per estrarre un archivio si usa l'opzione **`-x`** di **`tar`**:

```
tar -x -f NAME
```

Ad esempio, per estrarre nella directory corrente il contenuto dell'archivio **`etc.tar`** appena creato, si esegue il comando:

```
tar -x -f etc.tar
```

Modalità verbosa

(Si usa l'opzione **`-v`** di **`tar`**)

L'opzione **`-v`** di **`tar`** abilita la modalità verbosa, che stampa file e directory manipolati durante una operazione:

```
tar -v -f NAME <OPZIONI>
```

Ad esempio, per estrarre nella directory corrente in maniera verbosa il contenuto dell'archivio **`etc.tar`** appena creato, si esegue il comando:

```
tar -v -x -f etc.tar
```

Cambio di directory

(Si usa l'opzione **-C** di **tar**)

L'opzione **-C DIRECTORY** specifica la directory da cui **tar** preleva i file (creazione) o in cui tar scrive i file (estrazione):

```
tar -C DIRECTORY -f NAME <OPZIONI>
```

Ad esempio, per estrarre **etc.tar** in **/tmp**:

```
tar -C /tmp -f etc.tar -x
```

Ad esempio, per creare un archivio **tmp.tar** di **/tmp** (senza il prefisso **/tmp**) nella propria home:

```
tar -C /tmp -f tmp.tar -c .
```

Cave canem!

(L'opzione **`-C`** è subdola)

L'opzione **`-C`** ha affetto solo sulle opzioni ad essa successive; attenti a dove la mettete!

`tar -f tmp.tar -c . -C /tmp` effettua il backup della directory corrente (non di **`/tmp`**)…

L'opzione **`-C`** è inutile se specificate percorsi assoluti!

`tar -f tmp.tar -C /tmp -c /tmp` è un comando ridondante (l'opzione **`-C /tmp`** non serve)

Compressione di un archivio

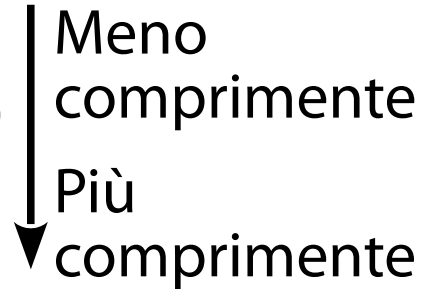
(Si usa una delle opzioni di compressione disponibili)

Il comando **`tar`** mette a disposizione alcune opzioni per la compressione dell'archivio tramite diversi algoritmi. Di seguito sono elencati gli algoritmi più comuni.

Opzione **`-z`** → Compressione GZIP (**`.gz`**)

Opzione **`-j`** → Compressione BZIP2 (**`.bz2`**)

Opzione **`-J`** → Compressione XZ (**`.xz`**)



Un esempio concreto

(Creazione ed estrazione di un archivio compresso di **`/etc`**)

Si vuole creare un archivio compresso GZIP di nome **`etc.tar.gz`** della directory **`/etc`**.

```
tar -z -f etc.tar.gz -c /etc
```

Si vuole espandere l'archivio compresso GZIP nella directory **`/tmp`**.

```
tar -z -f etc.tar.gz -C /tmp -x
```

Una osservazione strana

(Sollevabile solo da chi ha un minimo di esperienza in UNIX)

Gli studenti più ~~smanettoni~~ esperti[1] sanno che il comando **`tar`** (ed alcuni altri) può ricevere opzioni senza il trattino:

```
tar zcvf tmp.tar.gz /tmp
tar zxvf tmp.tar.gz
```

Che diavoleria è mai questa?

1 Che hanno installato almeno una volta un software da un archivio UNIX

Una risposta altrettanto strana

(Il motivo è la diaspora UNIX)

Il motivo di questa strana proprietà risiede nella **diaspora UNIX**.

Diaspora UNIX: è lo sviluppo indipendente di distribuzioni UNIX seguito a due eventi.

- La distribuzione del codice sorgente di UNIX System V R4.
- Lo studio dei ciclostili di John Lions in seguito al ritiro del codice sorgente con la distribuzione UNIX V7 nel 1979.

Ricordate il discorso delle subdole incompatibilità accennato nella seconda lezione?
Eccole, le subdole incompatibilità!

Alcune differenze tra distribuzioni UNIX

(So close, yet so far)

Distribuzione System V. I comandi offrono opzioni in formato breve (trattino più lettera).

Distribuzioni BSD. Alcuni comandi (**`tar`**, **`dd`**, **`ps`**) offrono opzioni senza trattino.

Distribuzione GNU. I comandi offrono opzioni in formato breve, lungo (due trattini più parola) e BSD.

Esercizio 24 (4 min.)

Create tre archivi della vostra home directory:

- **`/tmp/home.tar.gz`**
- **`/tmp/home.tar.bz2`**
- **`/tmp/home.tar.xz`**

Come variano le dimensioni degli archivi?

Il comando `7z`

(Gestisce archivi in formato 7-ZIP, e non solo)

Un'alternativa moderna a **`tar`** è il comando **`7z`**, che gestisce archivi nei formati seguenti:

- 7-ZIP
- TAR
- ZIP
- RAR
- …

Il formato di un comando **`7z`** è il seguente:

```
7z COMANDO <OPZIONI> ARCHIVE_NAME FD1 ... FDN
```

Creazione di un archivio

(Si usa il comando **a** di **7z**)

Per creare un archivio si usa il comando **a** di **7z**, che accetta un elenco di file e/o directory **FD1**, **FD2**, … **FDN**.

```
7z a ARCHIVE_NAME FD1 FD2 ... FDN
```

Ad esempio, per archiviare il contenuto della directory **~/.config** nell'archivio **config.7z**, si esegue il comando:

```
7z a config.7z ~/.config
```

Visione elenco file di un archivio

(Si usa il comando **`l`** di **`7z`**)

Per vedere l'elenco dei file e directory in un archivio si usa l'opzione **`l`** di **`7z`**:

```
7z l ARCHIVE_NAME
```

Ad esempio, per elencare il contenuto dell'archivio **`config.7z`** appena creato, si esegue il comando:

```
7z l config.7z
```

Estrazione di un archivio

(Si usa il comando **`x`** di **`7z`**)

Per estrarre un archivio si usa il comando **`x`** di **`7z`**:

```
7z x ARCHIVE_NAME
```

Ad esempio, per estrarre nella directory **`dir`** il contenuto dell'archivio **`config.7z`** appena creato, si eseguono i comandi seguenti:

```
mkdir dir
cd dir
7z x ~/config.7z
```

Opzioni aggiuntive

(Attivabili con l'opzione **-m** di **7z**)

L'opzione **-m** abilita opzioni aggiuntive di **7z**.

- Uso di algoritmi di compressione specifici.
- Impostazione del livello di compressione.
- Cifratura dell'archivio.
- …

Le opzioni sono troppe per poter essere descritte qui.
La pagina seguente del manuale utente (fornito con il pacchetto software **7z**) illustra le opzioni possibili:

/usr/share/doc/p7zip/DOC/MANUAL/cmdline/switches/method.htm

Cifratura di un archivio

(Si usano le opzioni **-mhe=on** e **-p** di **7z**)

Per cifrare un archivio in fase di creazione, si abilita la cifratura con l'opzione **-mhe=on** e si imposta la chiave di cifratura simmetrica con l'opzione **-p** di **7z**:

```
7z -mhe=on -pKEY a ARCHIVE_NAME FD1 ...
```

Ad esempio, per archiviare il contenuto della directory **~/.config** nell'archivio **config.7z**, cifrando l'archivio con la chiave **password**, si esegue il comando:

```
7z -mhe=on -ppassword a config.7z ~/.config
```

Le conseguenze della cifratura

(Viene chiesta la password ad ogni operazione)

Una volta cifrato l'archivio, ad ogni operazione successiva sarà richiesta la password (in modalità interattiva o batch tramite l'opzione **`-p`**).

Ad esempio, si provi ad elencare il contenuto dell'archivio **`config.7z`** appena creato:

```
7z l config.7z
```

Esercizio 25 (2 min.)

Create un archivio della vostra home directory:

`/home/studente/home.7z`

Cifrate l'archivio con la chiave simmetrica **`secret`**.

(Nella speranza che l'archivio non sia gigantesco) Estraete l'archivio nella directory **`/tmp`**.

BASH E FILE

Scenario e interrogativi

(Quali strumenti sono disponibili per la gestione dei file e directory in BASH?)

Scenario: l'utente vuole scoprire gli strumenti a disposizione di BASH per la gestione di file e directory.

Interrogativi:
Quali sono tali strumenti? Come funzionano?

Operatori unari su file

(Sono una marea)

BASH mette a disposizione tanti operatori unari per la verifica di proprietà su file. Di seguito sono presentati i principali.

`-d FILE` VERO se **`FILE`** esiste ed è una directory

`-e FILE` VERO se **`FILE`** esiste

`-f FILE` VERO se **`FILE`** esiste ed è un file regolare

`-h FILE` VERO se **`FILE`** esiste ed è un link simbolico

`-r FILE` VERO se **`FILE`** esiste ed è leggibile

`-w FILE` VERO se **`FILE`** esiste ed è scrivibile

`-x FILE` VERO se **`FILE`** esiste ed è eseguibile

Un esempio concreto

(Controllo di leggibilità da e scrivibilità su **`/etc/passwd`**)

Ad esempio, per scoprire se il file **`/etc/passwd`** sia leggibile o no, si esegue il test di leggibilità e si stampa il relativo codice di uscita:

```
test -r /etc/passwd
echo $?
```

`0` → **`/etc/passwd`** è leggibile

Per scoprire se il file **`/etc/passwd`** sia scrivibile o no:

```
test -w /etc/passwd
echo $?
```

`1` → **`/etc/passwd`** è non è scrivibile

Esercizio 26 (1 min.)

Testate la leggibilità e la scrivibilità del file seguente:

`/etc/shadow`

Riuscite ad accedere al file in qualche modo?

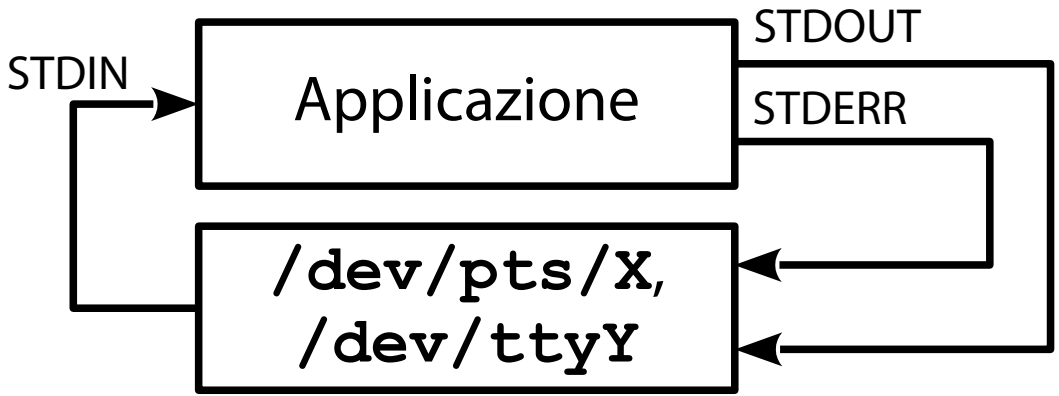
Canali di I/O
(STDIN, STDOUT, STDERR)

Ogni applicazione in esecuzione (BASH in primis) ha associati almeno tre **canali di I/O**.
STDIN. Usato per leggere gli input utente.
STDOUT. Usato per scrivere l'output.
STDERR. Usato per scrivere i messaggi di errore.

Associazione dei canali ad un file

(Di solito, il file del dispositivo terminale)

In condizioni normali, i tre canali sono associati al file speciale del dispositivo terminale connesso all'istanza di BASH in esecuzione.
Si legge dal terminale.
Si produce output sul terminale.
Si segnalano errori sul terminale.



Visione del file associato al canale

(Vengono messi a disposizione dei collegamenti simbolici)

I tre file seguenti sono collegamenti simbolici ai canali STDIN, STDOUT e STDERR dell'applicazione attualmente in esecuzione sul terminale corrente (di default, BASH):

`/dev/stdin`

`/dev/stdout`

`/dev/stderr`

I file associati ai canali possono essere risolti con il comando **`readlink -f`**.

Descrittore di file

(Un indice intero che punta al canale di I/O)

Un canale di I/O è accessibile tramite il suo **descrittore di file**.

Descrittore di file. È un numero intero rappresentante un indice ad un file aperto da una applicazione.

STDIN → Descrittore di file 0.
STDOUT → Descrittore di file 1.
STDERR → Descrittore di file 2.

Operatori di redirezione

(<, >, >>, &>)

L'operatore di redirezione **0< FILENAME** associa il canale STDIN ad un file. È permessa la sintassi alternativa **< FILENAME**.

```
cat < /etc/passwd
grep --color=yes -n -H root < /etc/passwd
```

Si noti il nome del file stampato da **grep** (standard input)!

Operatori di redirezione

(<, >, >>, &>)

L'operatore di redirezione **1> FILENAME** associa il canale STDOUT ad un file. È permessa la sintassi alternativa **> FILENAME**.

```
cat /etc/passwd > passwd
grep --color=yes -n -H root /etc/passwd > passwd
```

L'operatore di redirezione **2> FILENAME** associa il canale STDERR ad un file.

```
ls . nonesistente > out.txt 2> err.txt
```

Scarto dei messaggi di errore

(<, >, >>, &>)

Un uso produttivo dell'operatore **2>** consiste nello scarto dei messaggi di errore prodotti da una applicazione.
È sufficiente redirezionare STDERR sul file speciale di dispositivo **/dev/null**, che rappresenta una sorta di "cestino virtuale" (tutto ciò che riceve, lo scarta).

Ad esempio, per scartare gli errori prodotti da **ls**:

```
ls . nonesistente > out.txt 2> /dev/null
```

Operatori di redirezione

(<, >, >>, &>)

L'operatore di redirezione **>>** **FILENAME** appende l'output di un canale ad un file.

- **1>>** o **>>**: appende STDOUT
- **2>>**: appende STDERR

Esempi:

```
cat /etc/passwd > passwd-group
echo "----------" >> passwd-group
cat /etc/group >> passwd-group
```

Operatori di redirezione

(<, >, >>, **&>**)

L'operatore di redirezione **N>&M** copia il descrittore di file **N** nel descrittore di file **M**. D'ora in avanti, il canale puntato da **N** scrive sullo stesso file puntato da **M**.

2>&1 oppure **&>**: STDERR usa lo stesso file di STDOUT

Esempi:

```
ls . nonesistente > out-err.txt 2>&1
ls . nonesistente &> out-err.txt
```

Due trappole subdole

(La prima)

Il comando seguente scrive il valore 2 in un tunable del kernel, il che provoca la cancellazione di tutti i buffer di memoria memorizzati dal SO per motivi di efficienza. Tale operazione è possibile solo all'amministratore; pertanto, l'utente prova ad eseguire lo stesso comando tramite **`sudo`**, che eleva i privilegi a **`root`**:

```
sudo echo 2 > /proc/sys/vm/drop_caches
```

Nonostante **`sudo`**, il comando fallisce con un errore di "permesso negato" sul tunable. Perché?

Due trappole subdole

(La prima)

Le redirezioni sono applicate prima delle espansioni delle variabili e dell'esecuzione del comando.
Pertanto, la redirezione seguente:

```
> /proc/sys/vm/drop_caches
```

viene applicata prima dell'esecuzione del comando **`sudo`**, quando l'utente non ha ancora i privilegi di amministratore.

Due trappole subdole
(La seconda)

I due comandi seguenti, apparentemente equivalenti, non lo sono in realtà:

```
ls . nonesistente > out-err.txt 2>&1
ls . nonesistente 2>&1 > out-err.txt
```

Perché?

Due trappole subdole
(La seconda)

Le redirezioni sono applicate nell'ordine imposto dall'utente.

Il primo comando **`ls > out-err.txt 2>&1`**:

- redireziona STDOUT su **`out-err.txt`**;
- usa per STDERR lo stesso file di STDOUT.

Il primo comando **`ls 2>&1 > out-err.txt`**:

- usa per STDERR lo stesso file di STDOUT (il terminale);
- redireziona STDOUT su **`out-err.txt`**.

Esercizio 27 (2 min.)

Producete un elenco di tutti i file nel sistema con relativa dimensione:

```
/file1 dimensione1
/file2 dimensione2
...
```

Salvate l'output nel comando **`out.txt`**.
Scartate i messaggi di errore.

Gestione di file

(Si usa il builtin **`exec`**)

Il builtin **`exec`**, coadiuvato da opportuni operatori di redirezione, apre, redireziona, chiude file su descrittori arbitrari. È possible usare fino a 1024 descrittori diversi per ogni istanza di shell.

Ad esempio, per aprire il file **`/etc/passwd`** ed associarlo al descrittore di file 3, si esegue il comando:

```
exec 3< /etc/passwd
```

Gestione di file

(Si usa il builtin **`exec`**)

Ora è possibile leggere una riga di **`/etc/passwd`** con l'opzione **`-u`** del comando interno **`read`** (che specifica il descrittore di file da cui leggere):

```
read -u 3 line
```

Si stampa il contenuto della variabile **`line`**:

```
echo $line
```

Si ottiene la riga letta:

```
root:x:0:0::/root:/bin/bash
```

Gestione di file

(Si usa il builtin **`exec`**)

Si può anche aprire un file in scrittura:

```
exec 4> my-output.txt
```

Si scrive il contenuto della variabile **`line`** nel file appena aperto:

```
echo $line >&4
```

Si stampa il file:

```
cat my-output.txt
```

Si ottiene il contenuto di **`line`**:

```
root:x:0:0::/root:/bin/bash
```

Gestione di file

(Si usa il builtin **exec**)

Infine, si chiudono entrambi i file aperti:

```
exec 3<&-
exec 4>&-
```

Esercizio 28 (3 min.)

Usando i comandi interni **`exec`** e **`read`**, copiate le prime cinque righe di **`/etc/passwd`** nel file **`passwd-top5.txt`**.

Una riflessione sui comandi UNIX

(Nello specifico, i comandi di elaborazione del testo)

Si considerino i seguenti, semplicissimi, comandi:

```
ls
ls > ls.out
```

Il primo comando, il cui STDOUT è diretto al terminale:

- colora in modo diverso tipologie diverse di file;
- formatta l'output secondo la geometria del terminale.

Il secondo comando, il cui STDOUT è diretto su file:

- non effettua alcuna colorazione;
- stampa un file/directory per riga.

Una riflessione sui comandi UNIX

(Nello specifico, i comandi di elaborazione del testo)

Cosa se ne deduce?

I comandi sono progettati in modo tale da "capire" se i canali di I/O sono file oppure il terminale, e si comportano di conseguenza.

Terminale	→	Si abilitano le funzionalità interattive (colori, adeguamento dell'output alla geometria del terminale, indicatori di progresso, …).
File	→	Si disabilitano le funzionalità interattive. L'output è formattato per poter essere elaborato da altri programmi di manipolazione dei testi.

Filtri UNIX

(Trasformano un flusso dati in ingresso in un altro flusso dati in uscita)

I comandi UNIX per l'elaborazione dei testi sono concepiti come veri e propri **filtri**. Si parla, in tale contesto, di **filtri UNIX**.

Filtro: è una applicazione che trasforma un flusso di dati in ingresso (**input stream**) in un flusso di dati in uscita (**output stream**).

Il flusso di ingresso è letto dal canale STDIN.
Il flusso di uscita è scritto sul canale STDOUT.
Gli errori sono scritti sul canale STDERR.

Filtri UNIX

(Sono i "mattoncini LEGO" con cui costruire applicazioni più complesse)

I filtri UNIX possono essere in due modalità distinte.

Modalità standalone. Il singolo filtro è eseguito da solo, tipicamente in modalità interattiva (STDIN e STDOUT puntano al dispositivo terminale).

Si usa questa modalità quando si esplorano inizialmente le funzionalità offerte da un filtro.

Modalità combinata. L'output di un filtro è dato in pasto ad un filtro successivo.

È possibile combinare più filtri per creare nuove applicazioni!

Comunicazione tra i filtri: per il momento, tramite file temporanei (in futuro, con strumenti più elaborati).

Combinazione di filtri con redirezioni

(Si usano le redirezioni per creare file temporanei)

La soluzione più semplice per combinare filtri UNIX consiste nel memorizzare manualmente l'output di un comando in un file temporaneo, che sarà usato dal comando successivo.

Vantaggi.

Strategia semplicissima da implementare:

```
COMANDO1 > tmp.txt
COMANDO2 < tmp.txt
```

Svantaggi.

Richiede una gestione extra da parte dell'utente (scelta dei nomi dei file, cancellazione dei file dopo l'uso, …).
Soluzione poco elegante.

Un esempio concreto

(Un filtro complesso su **`/etc/passwd`**)

Si consideri la trasformazione su **`/etc/passwd`**:

- stampa delle colonne 1 e 7;
- stampa delle prime 10 righe.

Questa trasformazione può essere condotta in due passi (e un file intermedio) con i comandi **`cut`** e **`head`**.

Stampa delle colonne 1 e 7:

```
cut -d":" -f 1,7 /etc/passwd > tmp.txt
```

Stampa delle prime 10 righe:

```
head tmp.txt
```

Esercizio 29 (5 min.)

Individuate i dieci file più grandi all'interno della vostra home directory.

Espansione di percorsi

(Utilissima per identificare percorsi a partire da pattern)

BASH fornisce un meccanismo di **espansione dei percorsi** (**pathname expansion**). Tale espansione avviene tramite l'uso di opportuni caratteri speciali (**wildcard**) `?`, `*`, `[]` che consentono di definire modelli di file (**pattern**). I pattern principali sono:

`?`	→	un qualunque carattere
`*`	→	una qualunque sequenza di caratteri
`[`C_1C_2`...]`	→	i caratteri C_1, C_2, …
`[!`C_1C_2`...]`	→	NON i caratteri C_1, C_2, …
`[^`C_1C_2`...]`	→	NON i caratteri C_1, C_2, …
`[`C_1`-`C_2`]`	→	tutti i caratteri da a C_1 a C_2

Esempi di espansione di percorso

(Molto comodi)

Per elencare tutti i file che iniziano con la lettera **`a`**:

```
ls a*
```

Per elencare tutti i file che iniziano con un digit:

```
ls [0-9]*
```

Per elencare tutti i file che non iniziano con un digit:

```
ls [!0-9]*
```

Per elencare tutti i file di cinque lettere con radice **`file`**:

```
ls file?
```

Esempi di espansione di graffe con file

(Ancora più comodi)

Si crei un file vuoto **`file_old.txt`**:

```
touch file_old.txt
```

Per copiare **`file_old.txt`** in **`file_new.txt`**:

```
cp file_{old,new}.txt
```

Per generare efficientemente parametri riferiti a percorsi assoluti:

```
ls /bin/{ls,cat}
```

Pattern: quoting o no?

(Una questione spinosa)

Alcuni comandi sfruttano l'espansione dei parametri di BASH e sfruttano il quoting per leggere file contenenti caratteri speciali.

Classico esempio: il comando **`ls`**.

`ls a*` fa quello che pensate (visualizza i file che iniziano con la lettera **`a`**).

`ls 'a*'` non fa quello che pensate (visualizza i metadati del file di nome letterale **`a*`**).

`ls "a*"` non fa quello che pensate (visualizza i metadati del file di nome letterale **`a*`**).

Pattern: quoting o no?

(Una questione spinosa)

Altri comandi operano di default con pattern quotati.

`find /etc -name "*.conf"` fa quello che pensate (trova ricorsivamente tutti i file in **`/etc`** terminanti con l'estensione **`.conf`**.

`find /etc -name *.conf` potrebbe non fare quello che pensate.

Perché?

Le conseguenze del mancato quoting

(Non viene cercato ciò che viene imposto)

Si crei un file vuoto di nome **`a.conf`**:

```
touch a.conf
```

Si effettui la ricerca:

```
find /etc -name *.conf
```

Il comando **`find`** non trova alcun file. Ciò è in palese contrasto con il comando precedente, in cui sono stati individuati file di configurazione.

Che cosa è successo?

(Interviene l'espansione di bash)

Prima di eseguire il comando:

```
find /etc -name *.conf
```

BASH esegue una espansione dell'argomento non quotato **`*.conf`**.

L'espansione sostituisce il pattern **`*.conf`** con il file che lo soddisfa (**`a.conf`**).

In definitiva, il comando eseguito è:

```
find /etc -name a.conf
```

che non cerca i file desiderati.

Le conseguenze dell'imprecisione

(Non viene cercato ciò che viene imposto)

Si crei un altro file vuoto di nome **`a.conf`**:

```
touch b.conf
```

Si effettui la ricerca:

```
find /etc -name *.conf
```

Il comando **`find`** esce immediatamente con un errore. Anche ciò è in contrasto con quanto visto prima.

Che cosa è successo?

(Interviene l'espansione di bash)

Prima di eseguire il comando:

```
find /etc -name *.conf
```

BASH esegue una espansione dell'argomento non quotato **`*.conf`**.

L'espansione sostituisce il pattern **`*.conf`** con i due file che lo verificano (**`a.conf`** e **`b.conf`**).

In definitiva, il comando eseguito è:

```
find /etc -name a.conf b.conf
```

che è sbagliato sintatticamente.

Specifica di un pattern con il quoting

(Correzione dell'errore)

Eseguendo il comando con l'argomento tra apici, il comportamento è quello desiderato.

```
find /etc -name "*.conf"
find /etc -name '*.conf'
```

Ai fini dell'esecuzione, nell'esempio proposto l'uso di apici singoli (quoting forte) o doppi (quoting debole) è del tutto indifferente.

- Il quoting debole espande le variabili, quello forte no.
- Qui non sono presenti valori di variabili (ad esempio, **`$var`**) nel pattern.

Esercizio 30 (1 min.)

Individuate tutti i file in **`/etc`** che terminano con l'estensione **`.txt`**.