

Laboratorio di introduzione alla matematica computazionale
A.A. 2021/2022
01 - Introduzione

Fabio Durastante <fabio.durastante@unipi.it>
Sergio Steffè <steffe@mail.dm.unipi.it>

28 Settembre 2021 & 29 Settembre 2021

Obiettivi del corso

Il corso di **Laboratorio di introduzione alla matematica computazionale** serve a prendere familiarità con i **sistemi Linux**, e con alcuni **strumenti computazionali ed informatici** utili allo studio della matematica.

I **punti fondamentali** sono:

- ▶ Lavorare tramite terminale / shell su macchine multiutente.
- ▶ Operare tramite rete (connessioni SSH) su altri PC.
- ▶ Produrre e distribuzioni di documenti scientifici \LaTeX e HTML.
- ▶ Introduzione ad alcuni strumenti per il calcolo numerico e simbolico.

Organizzazione:

- ▶ Alcune lezioni plenarie (come questa).
- ▶ Sessioni di laboratorio in M-Lab & streaming.

Pagina su Moodle:

Gruppo A: <https://elearning.dm.unipi.it/enrol/index.php?id=351>

Gruppo B: <https://elearning.dm.unipi.it/enrol/index.php?id=352>

Organizzazione delle lezioni

- ▶ Tutte le lezioni saranno trasmesse in streaming su BBB; (o in caso di necessità su teams) potete ottenere il link alla stanza dalla pagina Moodle del vostro canale.
- ▶ Per i laboratori, ci sarà una divisione in due gruppi:
 - ▶ Gruppo A: Mercoledì 16:00-17:45 - Steffè
 - ▶ Gruppo B: Martedì 14:00-15:45 - Durastante
- ▶ Nella parte iniziale vi verranno fornite alcune informazioni pratiche;
- ▶ Le persone *in aula* lavoreranno in autonomia,
- ▶ Le persone *online* saranno divise in gruppi all'interno di stanze BBB in cui svolgeranno un esercizio (uno di voi condividerà lo schermo, e potrete discutere insieme della soluzione).
- ▶ Informazioni riguardanti i dettagli del corso e degli esercizi possono essere recuperate anche sulla pagina:
<https://people.cs.dm.unipi.it/limco/2021-22/>.

Struttura del computer

Facendo opportune semplificazioni, un computer fisico è composto da:

- ▶ Una alimentazione che fornisce le opportune tensioni ai componenti elettronici
- ▶ Una o più CPU, che si occupano di eseguire / interpretare delle istruzioni di un programma.
- ▶ Della memoria volatile (RAM), dove la CPU può memorizzare delle informazioni mentre è acceso.
- ▶ Della memoria non volatile in cui vengono memorizzate delle informazioni che non vanno perse allo spegnimento del computer (Dischi fissi SSD, etc..)
- ▶ Dei BUS, ovvero canali di comunicazione fra CPU / RAM / ecc. (tanti fili)
- ▶ Alcune altre periferiche con cui il computer si interfaccia e che servono per interagire col computer durante il funzionamento (tastiere, schermi, etc..)

Esistono comunque delle tecniche per emulare un computer fisico con un programma che gira su un altro computer: si parla allora di Macchine Virtuali (VM)

Sul computer si fa girare normalmente un programma detto Sistema operativo, che si occupa di gestire queste parti a “basso livello”.

Velocità dei computer

I primi strumenti di calcolo erano meccanici. Per un breve periodo furono usati computer analogici. Ora si usano quelli digitali.

Semplificando molto: sui fili dei computer digitali passano tensioni elettriche che variano in un certo range e sono considerati "1" o "0" se sono più grandi di un certo livello alto o più piccoli di un certo livello basso. Un segnale regolare, detto clock, e il fatto che l'elettronica è costruita in modo da non fare variare le tensioni tra "0" e "1" quando il clock è "1", permettono di assegnare il valore certo "0" o "1" al filo in certi precisi intervalli di tempo.

Per poter elaborare velocemente occorrono un clock alto (attualmente si va su alcuni Ghz), molti fili (attualmente bus a 64 fili sono comuni), e CPU potenti e numerose.

Detto ciò, è il software a fare la grossa differenza. In parte il sistema operativo, ma soprattutto gli algoritmi usati nel software. E' comune vedere che un modesto portatile con il software giusto batte sul tempo un gigantesco e potentissimo server da calcolo !

Sistemi operativi

La maggior parte dei computer disponibili in Università, con cui vi troverete ad interagire, utilizzano un sistema operativo Unix-like, in particolare qualche **distribuzione GNU/Linux**.

Un sistema operativo è composto (semplificando) da:

- ▶ Un **kernel**, che è un programma che gestisce l'hardware al livello più basso, e si occupa ad esempio di interagire con memoria, CPU, periferiche.
- ▶ Delle **applicazioni utente**, che si appoggiano al kernel per interagire con le varie funzionalità (ad esempio, utilizzare memoria e/o mostrare un'immagine a video).

Esempi: Windows è composto del kernel NT e l'interfaccia grafica, Mac OS X ha un kernel di nome Darwin, e sistemi come Ubuntu utilizzando Linux.

Sistemi Unix-like

- ▶ Unix è il successore del Multics che non fu proprio un successo. . .
- ▶ Alcuni sistemi sono più simili di altri;
- ▶ I sistemi Unix-like fanno parte di una famiglia che adotta delle convenzioni comuni. Sono storicamente stati usati nelle università, e per fare ricerca.
- ▶ Le distribuzioni GNU/Linux e Mac OS X sono esempi di tali sistemi.
- ▶ Windows, al contrario, adotta altri tipi di convenzioni.

Linux è nato in modo particolare, rispetto a sistemi commerciali come Windows e Mac OS X.

Linus Torvalds

Tutto è iniziato da un'email del 1991:



Linus Benedict Torvalds



Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-)

Linus (torv...@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-{.

Il progetto GNU

Quando Linux è nato, era solamente un kernel; qualche anno prima Richard Stallman aveva fondato la Free Software Foundation, con l'obiettivo di creare un sistema operativo **libero**.

Libero, in questo contesto, significa:

- ▶ Che chiunque lo può utilizzare senza pagare / doverlo acquistare.
- ▶ Che chiunque può leggere e modificare il codice sorgente, e redistribuire il risultato.
- ▶ Che chi migliora il software e lo redistribuisce è obbligato a condividere le proprie modifiche.

Nel 1991, al progetto GNU avevano un compilatore, delle applicazioni utente, ma gli mancava un kernel; da qui sono nati i sistemi GNU/Linux.

Distribuzioni GNU/Linux

Dato che il software GNU e Linux sono liberi, chiunque può distribuirli ed impacchettarli come preferisce:

- ▶ Un sistema Linux gira sulla maggior parte dei vostri telefoni.
- ▶ ... e probabilmente nella totalità dei vostri router ADSL, o nei sistemi multimediali delle auto moderne.
- ▶ Allo stesso tempo, è il sistema utilizzato da tutti i 500 supercomputer più potenti al mondo (<https://www.top500.org/>).

Poco dopo il primo rilascio di Linux, sono nate le prime distribuzioni Linux, come **Debian** e **Slackware**.

Ubuntu, una delle distribuzioni oggi più utilizzate come personal computer, è basata su **Debian**.

Licenze free software

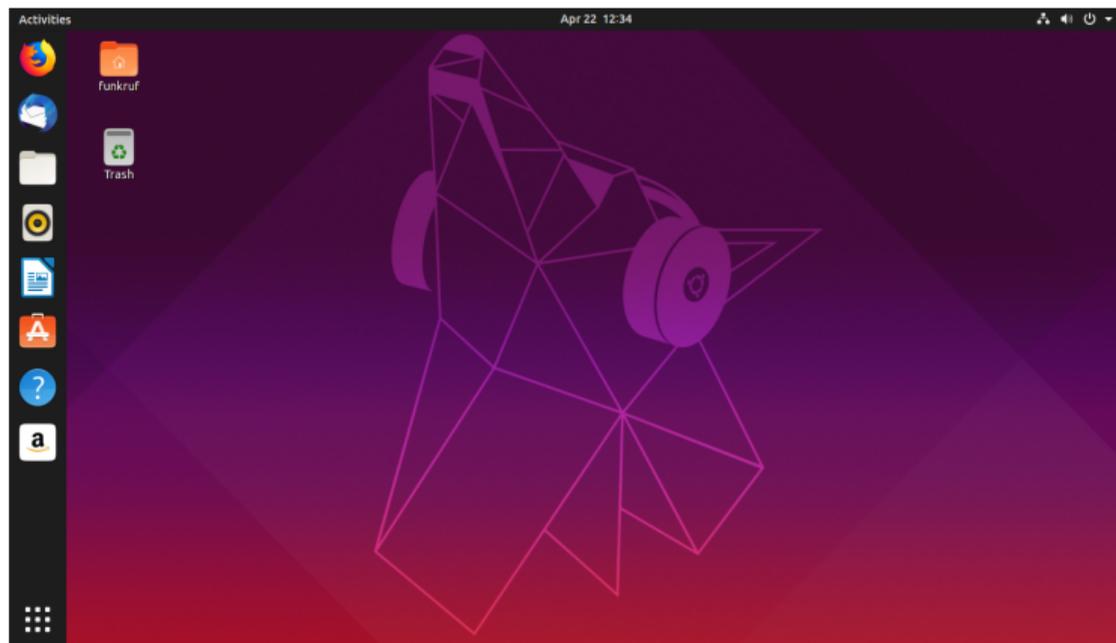
Il software che compone i sistemi GNU/Linux è perlopiù **free software**; ovvero è rilasciato liberamente, con più o meno vincoli per chi lo utilizza a fare lo stesso.

Questa strategia è sempre stata pratica comune in ambito accademico, ed in effetti molte licenze open source portano il nome di Università:

- ▶ License MIT / BSD (Berkeley), permettono uso praticamente incondizionato del software.
- ▶ Licenze GPL / LGPL invece richiedono a chi modifica il software di redistribuirlo sotto la stessa licenze, od una compatibile.

La shell

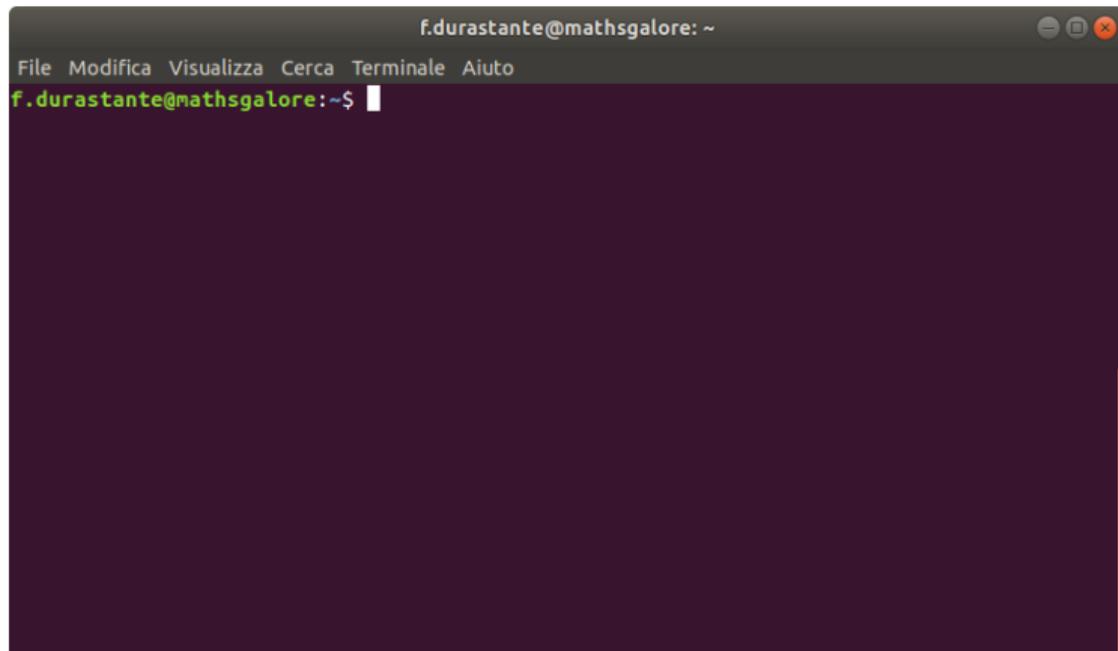
Oggi i sistemi operativi Linux forniscono molteplici interfacce grafiche:



ad esempio: GNOME (in figura), KDE, Xfce, etc.

La shell

Precedentemente, si poteva interagire con il sistema solo tramite una shell; questa è tutt'ora una parte importante del sistema:

A screenshot of a terminal window. The title bar at the top reads "f.durastante@mathsgalore: ~" and includes standard window control buttons (minimize, maximize, close). Below the title bar is a menu bar with the items "File", "Modifica", "Visualizza", "Cerca", "Terminale", and "Aiuto". The main area of the terminal has a dark purple background and displays the shell prompt "f.durastante@mathsgalore:~\$" in green text, followed by a white cursor. The terminal is otherwise empty.

Virtual console

Usualmente le distribuzioni GNU/Linux fornivano la console grafica sulla virtual console 7, mentre le console da 1 a 6 venivano usate per console testuali.

- ▶ Si può switchare da una console all'altra con la combinazione CTRL+ALT+Fn da tastiera, dove n è un intero da 1 a 12.
- ▶ Questo setup non è più mantenuto sulla maggior parte delle distribuzioni più recenti.
- ▶ Il copy&paste è possibile tra consolle virtuali ma non tra consolle virtuali e consolle grafiche.

Notazione

- ▶ D'ora in poi descriveremo alcuni comandi per interagire con il sistema tramite la shell.
- ▶ Questi verranno sempre preceduti dal simbolo \$, per sottolineare che vanno inserite al prompt utente, che nell'immagine precedente era:

```
f.durastante@mathsgalore:~$
```

- ▶ Ad esempio, supponiamo di voler usare il comando date, che fornisce data e ora attuali:

```
$ date  
dom 29 set 2019, 10.06.27, CEST
```

- ▶ Curiosità: esiste anche il comando cal!

Shell

La shell non è che un programma come tutti gli altri. In particolare, la shell comunemente installata sui sistemi con cui interagirete si chiama `bash` (che sta per Bourne Again SHell).

- ▶ Il nome viene dal fatto che sostituì un'altra shell, chiamata appunto Bourne shell.
- ▶ Non è l'unica opzione disponibile, e di norma ogni utente può scegliere di utilizzare una shell alternativa.
- ▶ Nella maggior parte dei casi, i comandi fondamentali sono simili.
- ▶ La shell è anche programmabile.

Utenti e gruppi

I sistemi Unix, da sempre utilizzati nelle Università, supportano diversi utenti e gruppi.

- ▶ Storicamente, molti utenti diversi interagivano con la stessa macchina, e bisognava fare in modo che ognuno potesse avere accesso ai suoi file, ma non al resto.
- ▶ I permessi sono regolati tramite l'appartenenza a dei **gruppi**.
- ▶ L'utente, insieme al nome della macchina, è visibile dalla shell, dove il prompt solitamente riporta:

```
f.durastante@mathsgalore:~$
```

- ▶ Il prompt può essere utilizzato per dare dei comandi, ad esempio:
who, id, groups, last
- ▶ Questi permettono di avere informazioni circa l'utente attuale, e quelli che stanno utilizzando la macchina.

Caratteristiche di un utente

Un utente è identificato da:

- ▶ Uno user id `uid`, che è solitamente un intero ≥ 1000 per utenti “umani”.
- ▶ Ci sono alcuni utenti di sistema, che hanno funzioni diverse, e solitamente `uid` < 1000 .
- ▶ Un gruppo principale, a sua volta identificato ad un intero (`gid`).
- ▶ L'appartenenza ad altri gruppi, con i loro `gid`.

Ad esempio, sulla macchina remota del corso posso utilizzare il comando `id`:

```
f.durastante@mathsgalore:~$ id
uid=1002(f.durastante) gid=1002(f.durastante) groups=1002(f.durastante),
27(sudo)
```

La lista dei gruppi a cui appartiene un utente si può stampare anche con il comando `groups utente`.

Il comando su

Una volta aperta una shell, è possibile cambiare utente utilizzando il comando su (switch user):

```
f.durastante@mathsgalore:~$ su d.lampa
Password:
d.lampa@mathsgalore:/home/d.lampa$
```

- ▶ La maggior parte degli utenti di sistema non hanno una password, ed è dunque impossibile loggarsi in questo modo.
- ▶ L'utente root può assumere qualunque identità senza bisogno di password.
- ▶ Utilizzando su senza argomenti, si può diventare utente root, se si conosce la password.

Sudo

Sui sistemi GNU/Linux più recenti si trova solitamente il comando `sudo`, che permette di eseguire singoli comandi come `root` oppure di diventare `root` a determinati utenti:

- ▶ Solitamente questi sono gli utenti nel gruppo `sudo`.
- ▶ Sulle macchine condivise, come quelle in Aula M-Lab o Aule 3 e 4, i vostri utenti non sono abilitati ad utilizzare `sudo`.
- ▶ Il comando `sudo -s` permette di aprire una shell come `root`.

I permessi di `root` sono necessari per:

- ▶ Installare software di sistema (ed effettuare aggiornamenti)
- ▶ Modificare (quasi) qualunque file fuori da `/home/mioutente/` e `/tmp/`.

L'utente root

Su ogni sistema Linux esiste un utente “speciale”, che è l'amministratore della macchina, e si chiama root; il suo uid è 0.

```
f.durastante@mathsgalore:~$ id root
uid=0(root) gid=0(root) groups=0(root)
```

- ▶ L'utente root può tutto sempre e comunque;
- ▶ Gli altri utenti devono rispettare alcune restrizioni, ad esempio circa i permessi dei file, oppure l'accesso a determinate periferiche.

Una particolarità di Linux, rispetto a Windows, è la necessità di avere un file system con un sistema di permessi¹.

¹Sebbene il file system più recente NTFS supporti dei permessi, questi non sono centrali al funzionamento del sistema. Al contrario, Linux non può funzionare su un file system che non gestisca i permessi.

Cambiare la password

Ogni utente può modificare la propria password con il comando `passwd`.

- ▶ Nel vostro caso la password che utilizzate è sincronizzata con quella delle vostre credenziali di ateneo!
- ▶ Per cambiare la password delle credenziali di ateneo si adopera invece l'apposito sito.
- ▶ È opportuno che la password non sia troppo debole, ovvero non il vostro nome o cognome, data di nascita, ecc.²
- ▶ Solo l'utente `root` può impostare una password per gli altri utenti.

²Questo permette di evitare che qualcuno la indovini tramite prove con dizionario, e usi il vostro account per inviare spam.

File system

Si parla di file system per indicare i modi in cui il sistema operativo memorizza e organizza i *file* di dati. Linux usa un file system virtuale, in cui sono anche "montati" i vari file systems che memorizzano i dati organizzati in files e directories sui cosiddetti "dispositivi a blocchi".

- ▶ Un disco (SSD, o con un piatto rotante), non conosce il concetto di file;
- ▶ Invece, permette di accedere ad un certo numero di "blocchi", ciascuno dei quali può essere utilizzato per memorizzare una certa sequenza di byte, e può essere indirizzato separatamente.
- ▶ Il sistema operativo utilizza questi blocchi in un modo standardizzato, per permettere all'utente di interfacciarsi al sistema accedendo a dei file.
- ▶ Il file system più usuale su macchine GNU/Linux è ext4. Ne esistono altri (btrfs, xfs, ...). Ad esempio MAC OS X utilizza HFS+ e Windows NTFS.

Struttura del file system

I file system su Linux funzionano nel seguente modo:

- ▶ Il file system alloca dello spazio sul disco per memorizzare una tabella di indirizzamento.
- ▶ In questa zona risiedono dei puntatori (`inode`), che contengono informazione su dove trovare un determinato file nel disco, ed altre informazioni (spazio occupato, permessi, ...).
- ▶ È possibile avere più puntatori che puntano allo stesso file: in questo caso si dice che si è creato un `hard-link`. È possibile anche creare collegamenti “simbolici”: sono dei puntatori speciali che puntano al percorso del file, e non all'`inode`.

Struttura del file system

I file all'interno di un file system sono organizzati in modo gerarchico, ad albero.

- ▶ Esiste un elemento radice, denotato da /
- ▶ Questo può avere dei figli, ovvero delle sottodirectory o file:
/file1 /file2 /dir1 /dir2
- ▶ Le directory sopra possono a loro volta avere dei figli:
/dir1/file3 /dir1/file4 /dir2/file5
- ▶ Il comando `cd` si può usare nella shell per cambiare la cartella di lavoro.
- ▶ I nomi speciali `.` e `..` indicano rispettivamente la cartella corrente e quella genitore.
- ▶ La struttura del file system è standardizzata, e l'elemento radice di un sistema operativo Linux contiene delle cartelle ben note.

Cartelle standard

All'interno del file system radice / si trovano le seguenti cartelle:

- ▶ /bin: contiene alcuni programmi eseguibili
- ▶ /boot: file necessary per il boot.
- ▶ /dev: file per accedere ai device³.
- ▶ /etc: file di configurazioni per programmi di sistema.
- ▶ /home: questa cartelle contiene delle sottocartelle con le directory personali degli utenti, e.g., /home/f.durastante.
- ▶ /lib: librerie di sistema.
- ▶ /root: La home dell'utente root.
- ▶ /proc, /sys: hanno un significato particolare – che vedremo in seguito.
- ▶ /tmp: per i file temporanei.
- ▶ /var: file che cambiano frequentemente, e.g., i log di sistema.

³Questa è una delle particolarità di Unix: tutto è un file, inclusi i dispositivi esterni

Il comando `ls`

Il comando `ls` si può utilizzare per ottenere una lista dei file all'interno di una directory.

Ad esempio:

```
$ ls /home/f.durastante  
boarding-pass.pdf  
Desktop  
Development  
Documents  
Downloads  
...
```

Come vedremo in laboratorio, esistono molte opzioni per ottenere più informazioni.

Permessi dei file

Con il comando `ls -l` possiamo ottenere informazioni più dettagliate circa i permessi di un determinato file. Proviamo con il file `boarding-pass.pdf`:

```
$ ls -l boarding-pass.pdf
-rw-rw-r-- 1 f.durastante f.durastante 191242 mag 11 15:38 ...
... boarding-pass.pdf
```

Il campo dei permessi è il primo, e le lettere hanno il seguente significato:

- ▶ Il primo campo è per degli attributi speciali (ad esempio, le directory hanno `d`).
- ▶ Seguono tre campi per eventuali permessi concessi all'utente proprietario: lettura (`r`), scrittura `w`, ed esecuzione (`x`).
- ▶ Si conclude con altri sei campi, tre per i permessi del gruppo (ancora `rw``x`) e i permessi per tutti gli altri.

Permessi delle directory

Proviamo a ripetere l'esempio con una directory:

```
drwxr-xr-x 4 f.durastante f.durastante 28672 set 26 15:12 Downloads
```

- ▶ Qui si nota l'attributo `x` per l'esecuzione.
- ▶ Nel caso delle directory, questo ha un significato particolare: chi può "eseguire" una directory ci può accedere.
- ▶ Possiamo utilizzare il comando `chmod` per cambiare i permessi.

Esempio:

```
f.durastante@mathsgalore:~$ chmod u-x Downloads/  
f.durastante@mathsgalore:~$ cd Downloads  
bash: cd: Downloads: Permission denied
```

- ▶ `chmod u-x` significa: toglì all'utente (`u`) il permesso di esecuzione.

Comandi potenzialmente utili

- ▶ `chown` cambia il proprietario di un file; è possibile solo per l'utente root!
- ▶ `chgrp` cambia il gruppo di un file.
- ▶ `chmod` altera i permessi ad un file.

I permessi si possono rappresentare anche in notazione "ottale". Dato che i campi `rxw` sono 3 valori binari, possiamo associarli un numero in base 2 come 010 per `-w-`, o 110 per `rw-`.

Un numero binario di 3 cifre corrisponde ad un intero da 0 a 7. Il comando `chmod` capisce anche questa notazione, per cui possiamo utilizzare:

```
$ chmod 777 boarding-pass.pdf
```

che permette l'accesso libero al file (permessi `rxw` per tutti).

Montare un filesystem

Ogni disco può avere uno o più file-system, contenuti in altrettante partizioni. Ognuna di queste può venire “montata” all’interno di una cartella.

Questo significa che il suo contenuto viene reso accessibile concatenando la sua radice con quella cartella.

Il disco di sistema viene sempre montato su /. Altri dischi vengono montati su sottocartelle (ad esempio, chiavette USB, dischi esterni, e simili).

Il comando per montare un disco è `mount`. Solo `root` può usarlo!

Un utente senza privilegi può utilizzarlo per controllare i file-system montati.

Ci sono però vari casi che esaminiamo in seguito in cui l’utente normale può montare dei filesystem (per esempio quando attacca una penna USB o monta remotamente una cartella con `sshfs`).

Mount

Proviamo a dare questo comando su un portatile:

```
$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=12149516k,
    nr_inodes=3037379,mode=755)
[...]
tmpfs on /run type tmpfs (rw,nosuid,noexec,relatime,
    size=2434428k,mode=755)
/dev/sda1 on / type ext4 (rw,relatime,errors=remount-ro,
    data=ordered)
```

- ▶ Possiamo concludere che se ispezioniamo la cartella `proc` vedremo la cartella “radice” del device virtuale `proc`.
- ▶ Unix segue la filosofia “everything is a file”. Ad esempio, `/dev/sda1` è la prima partizione del mio disco fisso, ma è anche visibile come un file.

Montare dischi esterni

Come abbiamo già detto, solo root può montare device addizionali.

- ▶ I sistemi Linux per computer personali prevedono comunque la possibilità di montare chiavette e dischi esterni come utenti “normali” .
- ▶ Il sistema può essere configurato per permettere ad alcuni utenti di chiedere il permesso di effettuare operazioni particolari.
- ▶ La maggior parte di queste operazioni di “richiesta di autorizzazione” viene effettuate tramite le applicazioni grafiche, e risulta piuttosto complessa da farsi attraverso la shell.

The Unix philosophy

Sistemi Unix e Unix-like sono stati sviluppati con le seguenti linee guida in mente:

- ▶ Scrivere software che faccia una sola cosa, e la faccia bene.
- ▶ Scrivere software facili da utilizzare insieme.
- ▶ Realizzare software che gestisca flussi di testo.

Come vedremo l'insieme di queste priorità ha portato allo sviluppo di alcuni strumenti semplici quanto efficaci.

Struttura di un comando

La maggior parte dei comandi su Linux segue convenzioni simili circa la sintassi da usare per specificare gli argomenti.

- ▶ Solitamente, un comando accetta uno o più argomenti che sono oggetto dell'azione.
- ▶ È possibile specificare delle opzioni che ne cambiano il comportamento; queste hanno quasi sempre una doppia sintassi, ad esempio:
 - ▶ `ls -a` indica che si vuole una lista dei file includendo quelli nascosti, e `-a` è l'opzione.
 - ▶ Lo stesso comportamento si può ottenere con `ls --all`.

In generale, un comando potrebbe avere il seguente aspetto:

```
$ mycommand --option1 --option2 value --option3=4 \  
-l -j -akm file1 file2 file3
```

La sintassi “-akm” è solitamente equivalente a “-a -k -m”.

Esempio

Come esempio, il comando `ls -lrt` è equivalente a `ls -l -r -t`, dove le opzioni indicano:

- ▶ `-l`: lista dettagliata.
- ▶ `-t`: lista ordinata per data di ultima modifica.
- ▶ `-r`: l'ordine della lista viene invertito (il file più recente sarà riportato in fondo).

In generale possiamo scoprire come utilizzare un comando utilizzando il comando `man`, che sta per manuale. Ad esempio: `man ls`.

Output di man ls

```
LS(1)                                User Commands                                LS(1)
NAME
ls - list directory contents
SYNOPSIS
ls [OPTION]... [FILE]...
DESCRIPTION
List information about the FILEs (the current direc-
tory by default).  Sort entries alphabetically if
none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for
short options too.

-a, --all
do not ignore entries starting with .

-A, --almost-all
do not list implied . and ..

--author
with -l, print the author of each file
[...]
```

Nome ed estensioni

In alcuni sistemi operativi i nomi e le estensioni dei file hanno un ruolo particolare. Ad esempio, un file di nome documento.pdf viene riconosciuto come un file PDF.

- ▶ Nonostante questa convenzione sia usualmente rispettata, non è obbligatoria.
- ▶ In particolare, i programmi eseguibili non devono avere per forza un'estensione particolare (ad es., in Windows sono riconoscibili dall'estensione .exe).
- ▶ Il tipo di un file è solitamente determinato dai suoi primi byte sul disco.
- ▶ Esiste un programma (chiamato *file*) per riconoscere i vari tipi di file.

Qualche esempio

Facciamo qualche prova:

```
$ file /bin/bash
```

```
/bin/bash: ELF 64-bit LSB shared object, x86-64,  
  version 1 (SYSV), dynamically linked, interpreter /lib64/ld,  
  for GNU/Linux 3.2.0, stripped
```

```
$ file boarding-pass.pdf
```

```
boarding-pass.pdf: PDF document, version 1.4
```

```
$ sudo file /dev/sda1
```

```
/dev/sda1: block special (8/1)
```

```
$ sudo file -s /dev/sda1
```

```
/dev/sda1: Linux rev 1.0 ext4 filesystem data,  
UUID=582d46cf-604a-4e21-ada5-ebabf2d2dd0e
```

Notiamo che è stato necessario utilizzare l'opzione `-s` per l'ultimo comando, perché `/dev/sda1` è un file speciale.

Comandi per operare sui file

Abbiamo già visto il comando `ls`, che permette di elencare i file in un cartella. Altri comandi utili sono:

- ▶ `pwd`: ci dice in che cartella ci troviamo; il nome deriva da **p**rint **w**orking **d**irectory.
- ▶ `cd`: ci permette di cambiare la cartella corrente (il nome viene da “**c**hange **d**irectory”).

Esempio:

```
f.durastante@mathsgalore:~$ pwd
/home/f.durastante
f.durastante@mathsgalore:~$ cd Downloads
f.durastante@mathsgalore:~/Downloads$ pwd
/home/f.durastante/Downloads
```

Il comando `cd` senza alcun argomento ci fa sempre tornare nella directory home (in questo caso `/home/f.durastante`).

I comandi `rm` e `cp`

Il comando `rm` permette di rimuovere un file:

```
$ rm [-r] file
```

- ▶ Senza alcuna opzione, può essere usato solo su file e non directory.
- ▶ Chiamato con l'opzione `-r`, viene eseguito in modo ricorsivo, e permette di eliminare una directory e tutti i file contenuti.
- ▶ A livello di file system, elimina uno o più inode.

Il comando `cp` permette di copiare un file:

```
$ cp [-r] sorgente destinazione
```

- ▶ Senza alcuna opzione, `cp` può copiare un singolo file.
- ▶ Se viene specificata l'opzione `-r`, allora viene fatta una copia ricorsiva, e `cp` può copiare delle directory.

Il comando `mv`

Il comando `mv` permette di spostare un file o una directory:

```
$ mv sorgente destinazione
```

- ▶ Se sorgente e destinazione si trovano sullo stesso file system, allora l'operazione corrisponde a spostare un inode. In particolare, è immediata e atomica⁴.
- ▶ In caso contrario, è equivalente ad una copia seguita da una rimozione della sorgente.

⁴ovvero non si può "interrompere a metà".

Comandi per creare file

Il comando `touch` può essere utilizzato per creare file.

```
$ touch file1 file2
```

In realtà, il suo scopo non è propriamente questo:

- ▶ Se un file non esiste, ne crea uno vuoto con il nome fornito.
- ▶ Se invece esiste, aggiorna solamente la data di ultima modifica a quella corrente (o ad un'altra, sfruttando opportuni parametri — vedere `man touch`).
- ▶ viene usato quando si programma in congiunzione col comando `make` per forzare una ricompilazione

Naturalmente per creare file di testo si usano di solito degli editor, e moltissimi altri programmi creano come uscita file in formati particolari.

Comandi per creare directory

Il comando `mkdir` si può utilizzare per creare directory:

```
$ mkdir /home/f.durastante/prova
```

- ▶ Con il parametro `-p`, è possibile creare anche tutte le directory genitore necessarie; ad esempio:

```
$ mkdir -p /home/f.durastante/prova/altra/ancora
```

equivale (supponendo che `/home/f.durastante/prova` non esista) alla sequenza di comandi:

```
$ mkdir /home/f.durastante/prova
```

```
$ mkdir /home/f.durastante/prova/altra
```

```
$ mkdir /home/f.durastante/prova/altra/ancora
```

Globber

La shell ci permette di utilizzare delle espressioni particolari che vengono automaticamente trasformate in base ai file presenti sul file system.

- ▶ Il simbolo * può essere utilizzato per identificare una qualunque sequenza di caratteri.
- ▶ Il simbolo ? seleziona un carattere qualunque.
- ▶ L'espressione [a-b] seleziona qualunque carattere nell'intervallo fra a e b. Ad esempio:
 - ▶ [0-9] seleziona tutte le cifre decimali.
 - ▶ [a-z] tutte le lettere minuscole.
- ▶ L'espressione {stringa1,stringa2,stringa3} viene espansa per tre volte (in questo esempio) sostituendo ognuna delle stringhe proposte.

Vediamo qualche esempio.

Globbering (esempi)

Supponiamo che la cartella corrente contenga questi file:

```
$ ls
altrofile  file1  file2  file3  filea  fileb  prova
```

Allora:

```
$ ls file?
file1  file2  file3  filea  fileb
```

```
$ ls file[1-5]
file1  file2  file3
```

```
$ ls file[a-z]
filea  fileb
```

Globbering (esempi, parte 2)

Supponiamo che la cartella corrente contenga questi file:

```
$ ls
altrofile  file1  file2  file3  filea  fileb  prova
```

Allora:

```
$ ls *file*
altrofile  file1  file2  file3  filea  fileb
```

```
$ ls file{1,3}
file1  file3
```

```
$ ls f*
file1  file2  file3  filea  fileb
```

Organizzare i file

Questo rende spesso operazioni su un grande numero di file più efficienti attraverso la shell che utilizzando un'interfaccia grafica.

Qualche esempio:

- ▶ Come cancellare tutti i file di testo in una cartella?

```
$ rm *.txt
```

- ▶ E se volessimo spostare tutti i nostri file musicali sulla nostra pennina USB, supponendo che questi siano suddivisi per cartelle con autore e album?

```
$ cd ~/Music
```

```
$ cp */*/*.{mp3,ogg,flac} /media/f.durastante/USB/
```

Spazio su disco

Utilizzando il comando `du`, possiamo scoprire quanto spazio occupa un certo file, oppure una directory. Ad esempio:

- ▶ Quanto occupa questa presentazione?

```
$ du ./lezione-30-09.pdf
1196 ./lezione-30-09.pdf
```

- ▶ Questo numero è in kilobytes, non sempre semplice da decifrare. Più spesso useremo il comando

```
$ du -sh lezione-30-09.pdf
1,2M ./lezione-30-09.pdf
```

- ▶ L'opzione `-h` sta per "human", e fornisce un output più facilmente comprensibile.
- ▶ L'opzione `-s` sta per summarize, e serve a fornire solo il totale per una directory selezionata.

```
$ du -sh .
3,2 M .
```

Spazio su disco

Altri comandi utili sono `df`, che permette di visualizzare lo spazio libero sui vari dischi montati:

```
$ df
File system      1K-blocchi      Usati Disponib.  Uso% Montato su
udev              1707388          0   1707388    0% /dev
tmpfs              346004          1712    344292    1% /run
/dev/sda5         225762876 110172296 104052712  52% /
[ ... ]
```

Anche qui, l'opzione `-h` è quasi fondamentale:

```
$ df -h
File system      Dim. Usati Dispon.  Uso% Montato su
udev              1,7G     0    1,7G    0% /dev
tmpfs              338M  1,7M    337M    1% /run
/dev/sda5         216G  106G    100G   52% /
[ ... ]
```

Processi

L'esecuzione di un programma viene gestito dal kernel tramite la creazione di un **processo**.

- ▶ Ogni processo ha un identificativo dato da un numero intero, chiamato PID (Process ID). Questo viene assegnato alla creazione del processo.
- ▶ Il comando `ps` permette di visualizzare la lista dei processi, con il loro PID e le risorse utilizzate (CPU e RAM).
- ▶ Il comando `kill` permette di terminare un processo, dato il suo PID.
- ▶ Spesso si può ricorrere al comando `killall`, che chiude tutti i processi che corrispondono ad un certo nome.
- ▶ un processo può generare altri processi; il comando `ps tree` mostra l'albero dei processi.

Proviamo a vedere un esempio di utilizzo di `ps` e `kill`.

Segnali

- ▶ Il comando `kill` funziona inviando un segnale ad un processo. Il segnale di default è `SIGTERM`, che gli chiede di terminare in maniera composta.
- ▶ Il segnale `SIGKILL` permette invece di terminare un processo senza dargli alcuna possibilità di reagire. Utile nel caso il processo sia bloccato e non risponda.
- ▶ Altri segnali utili sono `SIGSTOP` e `SIGCONT`, che permettono di mettere in pausa e poi riprendere un processo.

Esempio:

```
$ kill -SIGSTOP PID
```

```
$ kill -SIGCONT PID
```

In tutti i casi possiamo usare `killall`, se non siamo a conoscenza del PID del processo.

Nice level

Ad ogni processo viene associato un **nice level**, che è un intero compreso fra -20 e 19 .

- ▶ Più è basso il valore di nice, più il processo è importante, e gli viene assegnata una priorità maggiore rispetto agli altri che girano sulla macchina.
- ▶ Il livello di default è 0 .
- ▶ Si può specificare un livello diverso con il comando `nice`. Per esempio:

```
$ nice -n 10 gedit
```
- ▶ Un utente diverso da `root` può assegnare solo valori di nice più grandi di 0 , ed in genere solo aumentarli.
- ▶ Alcuni processi che sono “time critical” hanno un livello di nice < 0 . Per esempio, questo è vero per `pulseaudio`, che gestisce il sottosistema audio.

Top e Htop

Un modo molto pratico per visualizzare informazioni sui processi in tempo reale sono i software `top` e `htop`.

- ▶ `top` è un programma standard che si trova su qualunque sistema Unix (ad es. anche sul Mac).
- ▶ `htop` invece non è sempre installato, ma è molto più facile da usare e da interpretare.
- ▶ In entrambi i casi, è possibile utilizzarli per visualizzare il livello di nice, o per inviare segnali di kill.

Pipes

Una delle caratteristiche più importanti dei comandi su sistemi Unix, legata alla Unix philosophy, è la possibilità di utilizzare l'output di un comando come input di un altro.

- ▶ Per ottenere questo risultato, si utilizza il simbolo |, ad esempio:

```
$ ps aux | more
```

- ▶ `more` è un software detto pager, che permette di scorrere un testo lungo poco alla volta.
- ▶ È stato di fatto sostituito dal più raffinato `less`, il cui nome deriva probabilmente da “less is more”, ma che di fatto è molto più completo.

Less

- ▶ Less permette di navigare all'interno dell'output con i comandi `up` / `down` a `Page up` / `Page down`.
- ▶ Permette di fare delle ricerche utilizzando il comando `/`.
- ▶ Possiamo visualizzare (ad esempio) dei numeri di riga utilizzando `less -N`.
- ▶ Qualsiasi opzione può venire attivata dopo il lancio semplicemente digitandola. Nel dubbio conviene utilizzare il comando `h` per avere una lista di quelle disponibili.
- ▶ Si può utilizzare il comando `q` per uscire.
- ▶ Per alcuni comandi possiamo specificare un numero di ripetizioni. Ad esempio, possiamo chiedere di cercare la una stringa e poi saltare le prime 100 ripetizioni.

Filtrare e manipolare l'output

Poter usare l'output come input di altri software rende molto utili alcuni piccoli programmi che altrimenti non avrebbero quasi scopo:

- ▶ Il comando `tail` permette di visualizzare solo le ultime righe dell'input che gli viene fornito.
- ▶ In maniera simile, il comando `head` fornisce solo l'inizio.
- ▶ Il comando `cat` permette di concatenare il testo di più file, utilizzandolo con la sintassi

```
$ cat file1 file2 ... fileN
```
- ▶ Il comando `sort` permette di ordinare le righe rispetto a vari criteri.

Esempio: cosa mostra il seguente comando?

```
$ du -sh * | sort -h | tail -n 3
```

Presentiamo altri due comandi utili in combinazione con l'operazione di pipe:

- ▶ Il comando `sed` permette di trasformare il testo.
- ▶ Il nome viene da **S**tream **E**ditor.
- ▶ Esempio:

```
$ ps aux | sed "s/f.durastante/d.lampa/"
```

sostituisce la prima occorrenza di `f.durastante` su ogni riga con `d.lampa`.

- ▶ Si può usare la variante

```
$ ps aux | sed "s/f.durastante/d.lampa/g"
```

per sostituire qualunque occorrenza (e non solo la prima).

- ▶ Questo esempio mostra il sottocomando `s` di `sed`, ma ce ne sono molti altri; provate a visualizzare `man sed`.

Il comando `cut`

Possiamo usare il comando `cut` per estrarre alcune colonne da un testo formattato (ad esempio, l'output di `ps aux`).

- ▶ Il comando `cut -d ' ' -f4` estrae il quarto campo ottenuto dallo spezzettare l'output utilizzando il delimitatore `' '`.
- ▶ Esempio: come possiamo estrarre l'anno dall'output del comando `date`?
- ▶ P.S.: questo esempio non è particolarmente utile, perché il comando `date` permette di ottenere lo stesso risultato utilizzando `date +%Y`.

Redirigere l'input e l'output

- ▶ Ora che sappiamo manipolare flussi di testo, potremmo volerli salvare come passo intermedio, e/o caricarli da un file.

- ▶ Il simbolo < permette di utilizzare un file come **standard input**. Ad esempio:

```
$ cut -d ' ' -f4 < date.txt
```

- ▶ In maniera simile, possiamo salvare l'output con il comando >.

```
$ date > date.txt
```

Questo comando permette di redirigere lo **standard output**.

- ▶ Se utilizziamo il simbolo >> al posto di > allora il file verrà esteso invece che sovrascritto.

- ▶ I programmi possono scrivere anche sul cosiddetto **standard error**, utilizzato per segnalare errori. In pratica, non è distinguibile dallo standard output, ma va rediretto in modo separato con i comandi 2> o 2>>; ad esempio:

```
$ comando > output.txt 2> errori.txt
```

Variabili d'ambiente

All'interno delle terminale sono definite delle variabili, che contengono varie informazioni ed in particolare configurazioni.

- ▶ Possiamo visualizzarle con il comando `env`. Solitamente hanno un nome in caratteri maiuscoli.
- ▶ Per stamparne il valore, possiamo utilizzare il comando `echo`:

```
$ echo $HOME  
/home/f.durastante
```

- ▶ Possiamo definire nuove variabili con la sintassi:

```
$ MIAVAR=prova
```

- ▶ Se vogliamo che queste siano visibili anche ai sottoprocessi, allora dobbiamo “esportarle” con `export $MIAVAR`.

- ▶ I comandi possono venire combinati in

```
$ export MIAVAR=prova
```

- ▶ Per evitare ambiguità, si può utilizzare la sintassi `${MIAVAR}` per richiamare il valore di una variabile.

La variabile \$PATH

La variabile d'ambiente \$PATH controlla dove il sistema cerca i programmi che si richiamano da terminale.

- ▶ È una lista di percorsi separati da `..`

- ▶ Ad esempio:

```
$ echo ${PATH}
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin\  
:/usr/games:/usr/local/games:/snap/bin
```

- ▶ Possiamo modificarla nel caso volessimo installare programmi aggiuntivi nella nostra home. In quel caso, potremmo specificare (ad esempio):

```
$ export PATH=$PATH:/home/f.durastante/bin
```

- ▶ Se vogliamo rendere questa modifica persistente, possiamo aggiungere una linea al file `.profile`.

Il comando grep

Un altro comando incredibilmente utile è `grep`, che ci permette di selezionare solo le linee che matchano un determinato pattern.

- ▶ Esempio:

```
$ ps aux | grep f.durastante
```

estrarrà solamente le linee che contengono la stringa `f.durastante`.

- ▶ Si può invertire il match (cercando ad esempio tutti i processi che non matchano `f.durastante`) utilizzando l'opzione `-v`.
- ▶ Si può utilizzare in maniera ricorsiva su dei file, ad esempio:

```
$ grep ciao . -R
```

cerca la stringa `ciao` in tutti i file in questa directory e nelle sue sottodirectory.

Il comando find

Il comando `find` si può utilizzare, come il nome suggerisce, per cercare file corrispondenti ad un determinato pattern all'interno

- ▶ Se vogliamo trovare i file il cui nome corrisponda ad un certo pattern utilizziamo, ad esempio

```
$ find . -name test\*
```

L'utilizzo di `*` per evitare che la shell espanda il simbolo `*`.

- ▶ Altri esempi:

```
$ find . -maxdepth 1 -name test\*
```

```
$ find . -atime 2 -maxdepth 1
```

Il secondo comando lista tutti i file a cui si è acceduto al più 2 giorni fa, e che si trovino in questa cartella.

- ▶ Esiste un'infinità di opzioni disponibili, potete consultare `man find`.

Script

Se utilizziamo spesso una sequenza di comandi, possiamo realizzare uno script da poter lanciare.

- ▶ Uno script altro non è che un file di testo che abbiamo reso eseguibile (`chmod a+x miofile.sh`)
- ▶ La prima linea deve contenere i caratteri speciali `#!`, che specificano il programma da usare per eseguire il file. Nel caso di un normal script che esegua comandi da terminale, questo è `#!/bin/bash` (**shabang** : SHArp bang o haSH bang).
- ▶ Possiamo eseguirlo con `./miofile.sh`, oppure metterlo in una cartella all'interno del `$PATH`.
- ▶ L'estensione `.sh` non è strettamente necessaria, anche se comune.

Editor di testo

Per editare file di testo, avremo bisogno di un editor. Ci sono svariate possibilità:

- ▶ Gedit, editor predefinito di GNOME (il desktop environment che troverete su Ubuntu). Facile e intuitivo, ma disponibile solo in modalità grafica.
- ▶ Nano, un editor molto basilare che è possibile utilizzare da terminale. Gli unici comandi da tenere a mente sono:
 - ▶ CTRL+O per salvare.
 - ▶ CTRL+X per uscire.
- ▶ vim ed emacs, utilizzabili in modalità grafica e da terminale, sono molto più potenti ma anche potenzialmente complessi da usare⁵.

⁵Ad esempio, vi accorgete che non è intuitivo chiudere nessuno dei due. . .

Riassunto

- ▶ Abbiamo visto come interagire con il sistema tramite terminale, e come funzionano utenti, filesystem, e processi.
- ▶ Molti dei comandi che abbiamo visto verranno approfonditi in pratica in laboratorio e in streaming, a partire dalla prossime settimane.

Crediti

Queste slide sono basate sulle slide per il corso “Laboratorio di comunicazione mediante calcolatore” di L. Robol e S. Steffè, A.A. 2020-2021.