

1. Linux

Concetti Fondamentali e Strumenti Essenziali

Indice

- Linux
- Utente root
- Gruppi e Password
- Shell Bash
- Comandi utili
- Comandi legati al file system
- Standard di input/output/error
- Pipeline
- Lista
- Operatori logici

Linux

Linux è un sistema operativo **multiutente** dove differenti utenti possono avere accesso al sistema avendo i propri dati, i propri programmi e impostazioni completamente separate da quelle di altri utenti oltre ad avere la possibilità di accedere alle risorse del sistema simultaneamente.

- L'operazione di **autenticazione** dell'utente, tramite **nome utente (username)** e **password**, che permette l'accesso alle risorse del sistema è detta **login**.
- Ad ogni **username** assegnato dall'amministratore del sistema corrisponde uno **user-id (UID)** assegnato dal sistema.

Il **login** può essere eseguito in ambienti diversi, **grafici** o **testuali** da **locale** o da **remoto**. L'**autenticazione** verifica che l'utente abbia i **requisiti** per accedere al sistema o ad un suo servizio e metterlo in condizione di interagire con la macchina.



Utente root/1

L'utente **root** (**superuser**, **administrator**) è un utente **privilegiato**. Ad esso sono riservati i compiti di **gestione** e **configurazione** della macchina.

L'utente **root** ha **poteri assoluti** sul sistema. Esso può:

- **aggiungere, eliminare e modificare** account degli utenti
- **installare** e *configurare* servizi
- **aggiungere e modificare** il file system
- **distruggere** tutto con un solo comando.



Utente root/2

1. Usare il comando ``su``

Il comando ``su`` (substitute user) permette di cambiare utente in una sessione di terminale. Se eseguito senza argomenti, cambia all'utente root.

```
bash
su -
```

Copia codice

Dopo aver eseguito questo comando, ti verrà richiesta la password dell'utente root. Una volta inserita correttamente, la tua shell sarà quella dell'utente root.

2. Usare ``sudo`` per eseguire comandi con privilegi di root

Il comando ``sudo`` permette agli utenti autorizzati di eseguire comandi come root senza dover cambiare utente.

```
bash
sudo command
```

Copia codice

Gruppi e Password

Gruppi

- Ogni utente può far parte di **uno o più gruppi**, definiti dall'amministratore del sistema.
- Ogni **gruppo** è identificato da un **group name** associato ad **un group-id (GID)** numerico.

Password

- Ogni utente può avere (e in seguito modificare) una propria **password**.
- La password non viene visualizzata sullo schermo: è personale e non deve essere rivelata.

ATTENZIONE !

Il sistema operativo Linux **distingue** tra lettere **maiuscole** e **minuscole** e pertanto:

- *username: admin*
- *username: Admin*
- *username: ADMIN*

NON sono lo stesso utente.

Come anche i seguenti file:

- *prova.dat* *PROVA.dat*
- *Prova.dat* *PROVA.DAT*

NON rappresentano lo stesso file.

Shell/1

La shell (**conchiglia**) è un **programma di sistema** che agisce da **interfaccia** tra **utente** e il **sistema operativo**.

E' un interprete che **legge** ed **esegue** le **istruzioni di comando** per:

- la creazione e l'avvio di processi;
- per la gestione di I/O, della memoria di massa e della memoria principale,
- per definire le politiche di protezione dei file e della comunicazione di rete.

Sebbene i sistemi **Linux** abbiano un'interfaccia utente grafica, la maggior parte dei **programmatori** e **utenti esperti** preferisce un'interfaccia a **linea di comando**, più veloce e potente, facilmente espandibile che permette all'utente di non dover sempre usare il mouse.

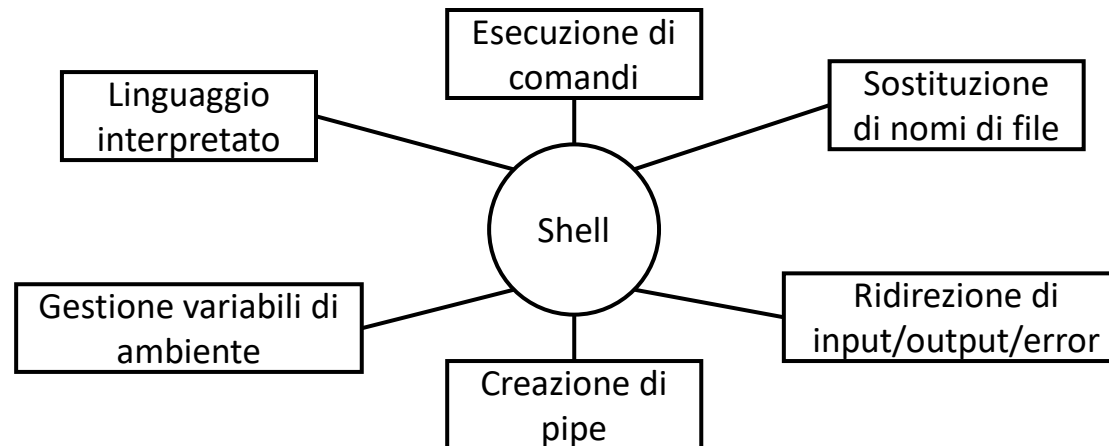
Shell/2

Quando la **shell** viene avviata:

- vi presenta il **prompt** (spesso un **simbolo di percentuale** o di **dollaro**, che permette di inserire comandi)
- **interpreta** il comando inserito
- **crea e avvia** il processo per la sua **esecuzione**.

La **shell** è programmabile, cioè permette di definire gli **script**.

- Uno **script** è un **programma** in **formato testuale** che racchiude **comandi** fondamentali per l'uso e l'amministrazione di un S.O.



Tipi di Shell

Esistono diverse shell, tra cui:

- Bourne shell (**sh**)
- Korn shell (**ksh**)
- C shell (**csh** ed il suo successore **tcsh**)
- Bourne again shell (**bash**)

Ci concentreremo su **bash**, la shell (quasi) standard in ambiente **Linux**.

Comandi

I **comandi** rappresentano una **richiesta di esecuzione**.

Vanno scritti rispettando **maiuscole**, **minuscole** e **spazi**, altrimenti non verranno riconosciuti, cioè risulteranno “not found”.

Forma generale di un comando:

comando [opzioni] [argomenti]

Opzioni:

- modificano l'azione del comando
- iniziano (quasi sempre) con un -

Argomenti:

- indicano l'oggetto su cui eseguire il comando.

Comando: pwd

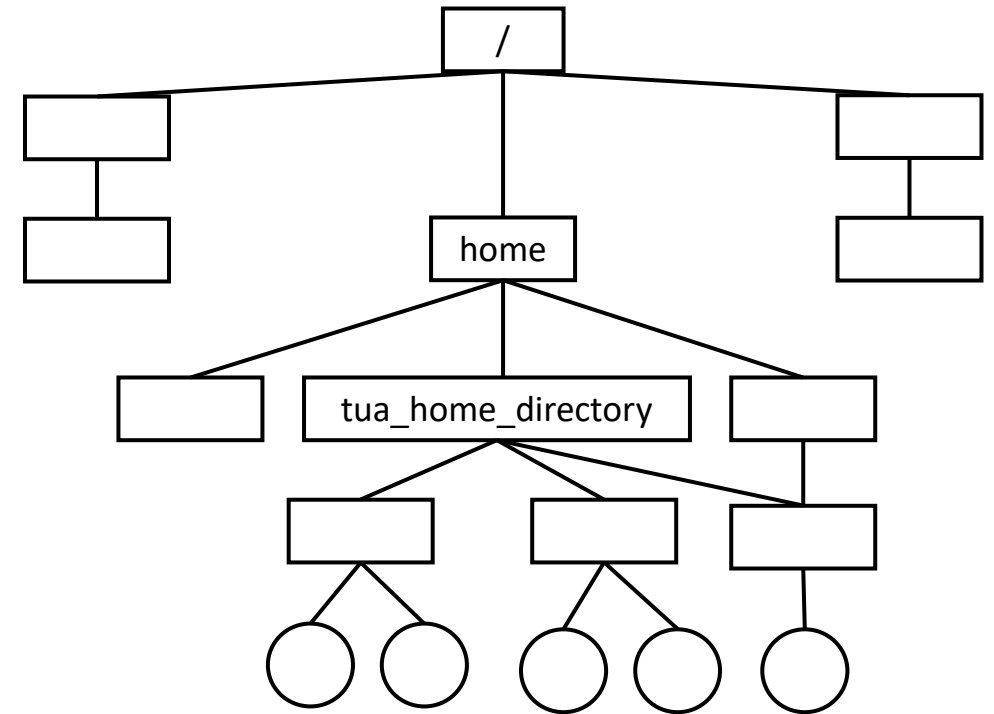
Per verificare quale sia la **directory corrente**, si utilizza il comando **pwd** (**P**rint **W**orking **D**irectory).

Esempio:

\$ pwd

Output:

/home/tua_home_directory



Comando: passwd

Per **cambiare** la propria **password** si usa il comando **passwd**.

Esempio:

\$ passwd

Output:

Changing password for [current user]

Current password: [insert current password]

New password: [insert the new password]

Retype new password: [retype the new password]

Passwd: all authentication tokens updated successfully.

Comando: man/1

Per aprire il **manuale** (**help on line**) riguardante un **comando** si deve digitare il comando **man** seguito dal comando da analizzare.

Esempio:

```
$ man passwd
```

Navigazione del manuale:

- **space (barra spaziatrice):** va avanti di una pagina
- **return (invio):** va avanti di una linea
- **q:** esce dal file

***NOTA:** Utilizzare **\$ sudo mandb** per aggiornare il database dei manuali.*

Comando: man/2

Per ottenere tutte le **pagine di un manuale** che contengono una determinata **parola chiave** si usa il comando **man -k** seguito dalla parola chiave ricercata.

Esempio:

```
$ man -k password
```

Output:

Elenco di tutte le pagine manuale che contengono la parola chiave specificata.

Comando: whatis

Per ottenere una **breve descrizione** dei **comandi di sistema** si usa il comando **whatis**.

Esempio:

```
$ whatis pwd
```

Output:

```
pwd (1)           - print name of current/working directory
```


Chi sono e chi è collegato?/1

Il comando per conoscere con quale identità si sta operando è:

\$ whoami

Il comando per visualizzare lo UserID dell'utente corrente è:

\$ id

Il comando per visualizzare lo UserID dell'utente specificato è:

\$ id *nome_utente*

Il comando per visualizzare tutti gli utenti loggati nel sistema è:

\$ who

Chi sono e chi è collegato?/2

Il comando per visualizzare informazioni dell'utente corrente è:

\$ finger -l

(se il comando non viene trovato è possibile installarlo mediante il comando: **sudo apt install finger**)

Il comando per visualizzare informazioni dell'utente specificato è:

\$ finger *nome_utente*

Il comando per visualizzare ed elencare gli ultimi utenti connessi è:

\$ last

Comando: uname

Per conoscere il nome del kernel del **Sistema Operativo** bisogna digitare il seguente comando:

```
$ uname
```

Per visualizzare tutte le informazioni relative al **Sistema Operativo** in uso, occorre digitare il comando:

```
$ uname -a
```

Creazione di un file

I file possono essere creati con un qualsiasi editor e quindi salvati.

Il comando **nano** è un editor di testo a riga di comando semplice e facile da usare disponibile su molti sistemi Unix e Linux.

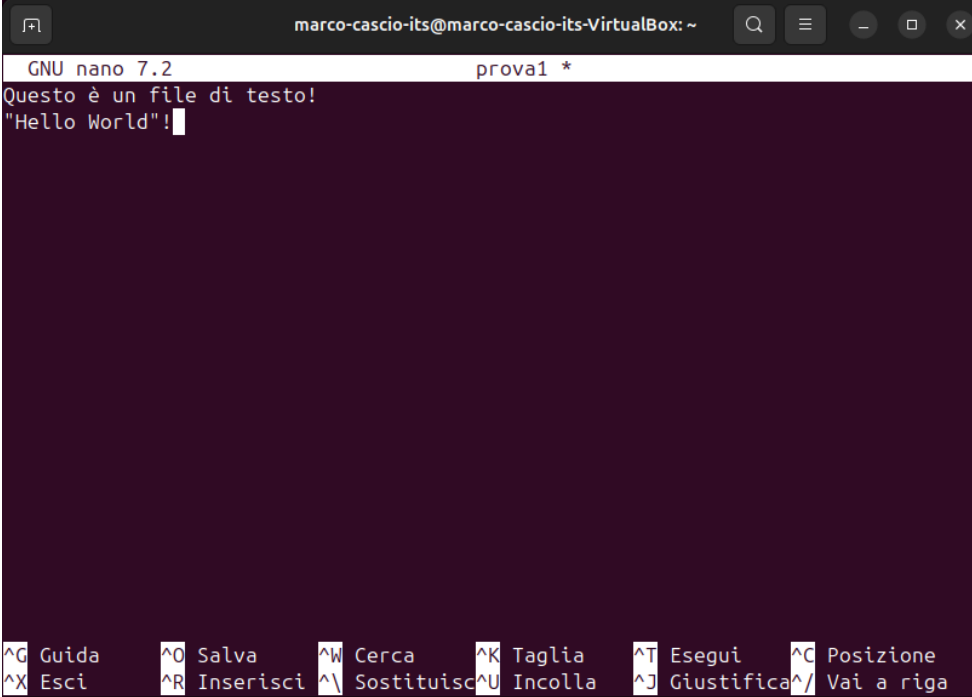
È particolarmente utile per **modificare file di configurazione** o **creare e modificare file di testo** direttamente dal terminale.

Esempio:

```
$ nano prova1.txt
```

Sequenze di controlli utili:

- **CTRL+X** per uscire dall'editor + **S** per salvare il file + **INVIO** per confermare il salvataggio del file.
- **CTRL+O** per salvare il file + **INVIO** per confermare il salvataggio del file.



```
marco-cascio-its@marco-cascio-its-VirtualBox: ~
GNU nano 7.2 prova1 *
Questo è un file di testo!
"Hello World!"
^G Guida  ^O Salva  ^W Cerca  ^K Taglia  ^T Esegui  ^C Posizione
^X Esci   ^R Inserisci  ^_ Sostituisci  ^U Incolla  ^J Giustifica  ^_ Vai a riga
```

Modifica di un file

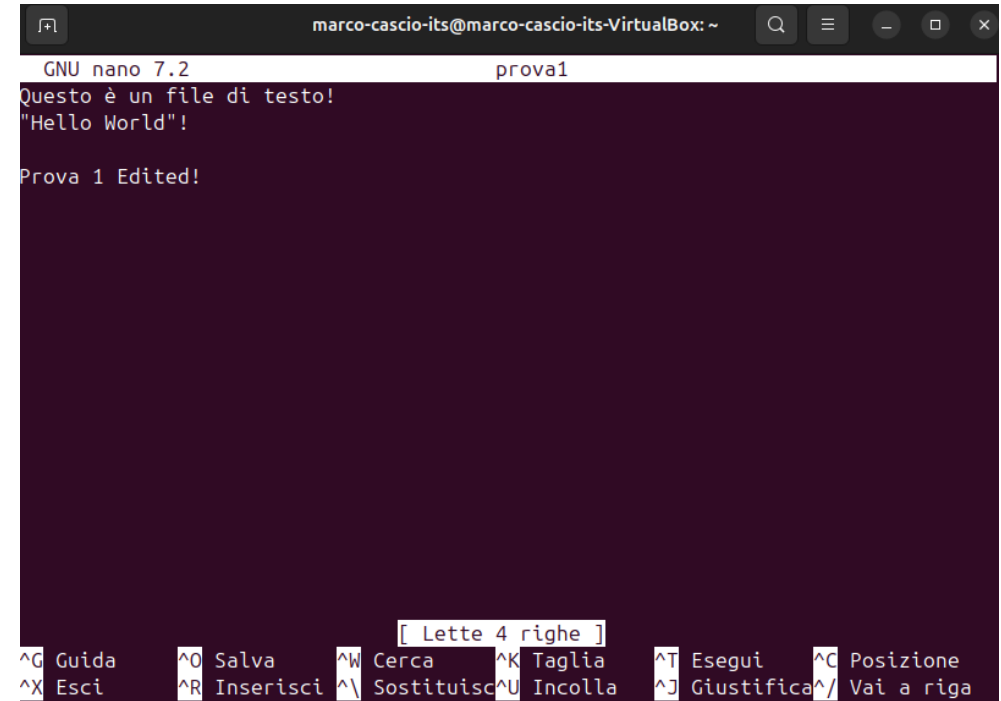
Il comando **nano** consente anche di modificare un file di testo esistente.

Esempio:

```
$ nano prova1.txt
```

Modificare il file di testo:

- Usare le **frecce direzionali** per spostarsi tra le righe.



```
marco-cascio-its@marco-cascio-its-VirtualBox: ~
GNU nano 7.2      prova1
Questo è un file di testo!
Hello World!
Prova 1 Edited!

[ Lette 4 righe ]
^G Guida  ^O Salva  ^W Cerca  ^K Taglia  ^T Esegui  ^C Posizione
^X Esci   ^R Inserisci ^\ Sostituisci ^U Incolla ^J Giustifica ^_ Vai a riga
```

Comando: Touch

Il comando **\$ touch *filename***

- se il file non esiste, crea un file vuoto
- se il file esiste, aggiorna la data e l'ora dell'ultimo accesso al file

Esempio:

\$ touch *prova2.txt*

- opera su singolo file

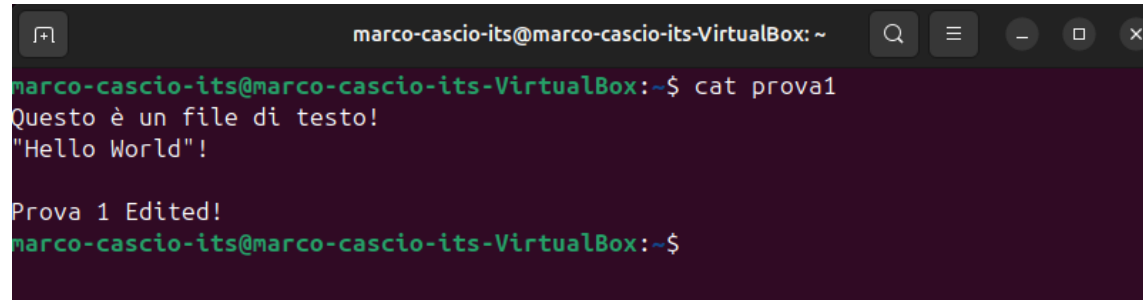
\$ touch *prova3.txt prova4.txt prova5.txt*

- opera su file multipli

Visualizzare il contenuto di un file

Per visualizzare il **contenuto** di un **file leggibile** (non binario) si possono utilizzare i seguenti comandi:

- `cat`
- `more`
- `most`
- `tail`
- `head`



```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ cat prova1  
Questo è un file di testo!  
"Hello World!"  
  
Prova 1 Edited!  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Comando: cat

- **cat**: mostra il **contenuto** del file facendolo **scorrere completamente** sullo schermo.

Esempio:

```
$ cat prova1.txt
```

```
$ cat cenerentola.txt
```

(il file *cenerentola.txt* è da scaricare e inserire in */home/tua_home_directory/disney*)

N.B. Completamento automatico:

È un modo attraverso il quale la shell aiuta l'utente a completare un comando.

Durante la digitazione di un nome o di un comando, premendo il tasto Tab, si chiede alla shell di provare a completarlo.

Esempio: \$ cat cene[Tab] rentola

I tasti ↑ ↓ richiamano i comandi usati in precedenza.

Comando: more

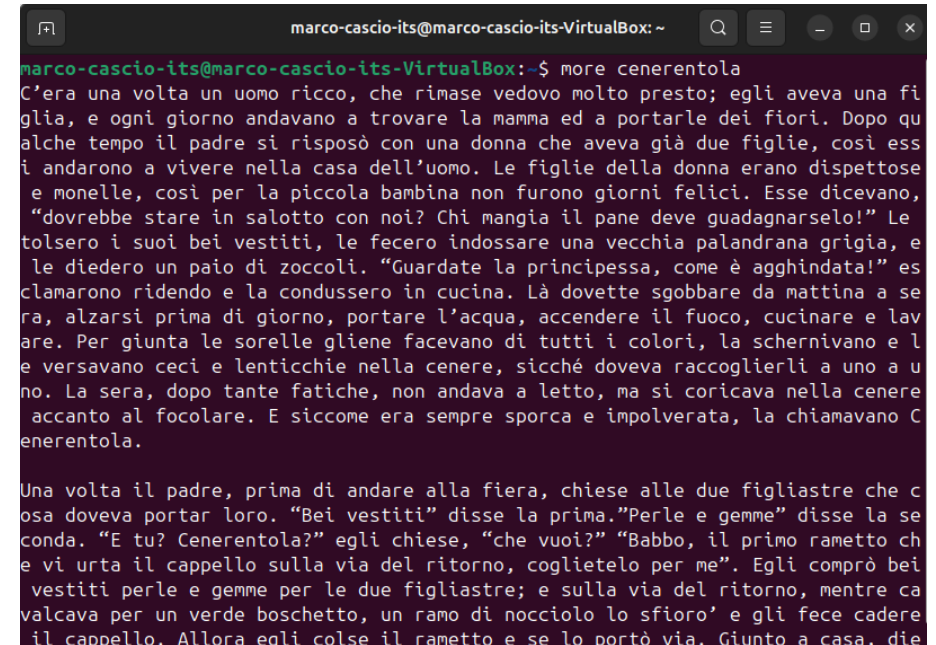
- **more**: mostra il contenuto del file, una pagina per volta.

Esempio:

```
$ more cenerentola.txt
```

Navigazione:

- **space (barra spaziatrice)**: va avanti di una pagina
- **return (invio)**: va avanti di una linea
- **q**: esce dal file



```
marco-cascio-its@marco-cascio-its-VirtualBox: ~$ more cenerentola
C'era una volta un uomo ricco, che rimase vedovo molto presto; egli aveva una fi
glia, e ogni giorno andavano a trovare la mamma ed a portarle dei fiori. Dopo qu
alche tempo il padre si risposò con una donna che aveva già due figlie, così ess
i andarono a vivere nella casa dell'uomo. Le figlie della donna erano dispettose
e monelle, così per la piccola bambina non furono giorni felici. Esse dicevano,
"dovrebbe stare in salotto con noi? Chi mangia il pane deve guadagnarselo!" Le
tolsero i suoi bei vestiti, le fecero indossare una vecchia palandrana grigia, e
le diedero un paio di zoccoli. "Guardate la principessa, come è agghindata!" es
clamarono ridendo e la condussero in cucina. Là dovette sgobbare da mattina a se
ra, alzarsi prima di giorno, portare l'acqua, accendere il fuoco, cucinare e lav
are. Per giunta le sorelle gliene facevano di tutti i colori, la schernivano e l
e versavano ceci e lenticchie nella cenere, sicché doveva raccogliarli a uno a u
no. La sera, dopo tante fatiche, non andava a letto, ma si coricava nella cenere
accanto al focolare. E siccome era sempre sporca e impolverata, la chiamavano C
enerentola.

Una volta il padre, prima di andare alla fiera, chiese alle due figliastre che c
osa doveva portar loro. "Bei vestiti" disse la prima. "Perle e gemme" disse la se
conda. "E tu? Cenerentola?" egli chiese, "che vuoi?" "Babbo, il primo rametto ch
e vi urta il cappello sulla via del ritorno, coglietelo per me". Egli comprò bei
vestiti perle e gemme per le due figliastre; e sulla via del ritorno, mentre ca
lalcava per un verde boschetto, un ramo di nocciolo lo sfiorò e gli fece cadere
il cappello. Allora egli colse il rametto e se lo portò via. Giunto a casa, die
```

Comandi: tail , head

- **tail**: mostra di **default** le **ultime 10 linee** di un file.

Esempio: **\$ tail cenerentola.txt**

- **head**: mostra di **default** le **prime 10 linee** di un file.

Esempio: **\$ head cenerentola.txt**

- Per entrambi i comandi l'opzione **-n** mostra le **prime** o le **ultime n** linee di un file.

Esempio:

\$ head -1 cenerentola.txt

- mostra la prima linea del file cenerentola

Esempio:

\$ tail -2 cenerentola.txt

- mostra le ultime 2 linee del file cenerentola

Comando: clear

Il comando **clear** consente di «pulire» la console.

Esempio:

\$ clear

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
asero, una a destra, l'altra a sinistra. Quando stavano per essere celebrate le  
nozze, arrivarono le sorellastre, che volevano ingraziarsi Cenerentola e parteci  
pare alla sua fortuna. Quando la coppia entrò in chiesa, la maggiore si trovò al  
la destra di Cenerentola, la minore alla sua sinistra. Allora le colombe cavarono  
loro un occhio a testa. Poi, all'uscita, la maggiore era a sinistra e la più p  
iccola a destra e le colombe cavarono l'occhio che era rimasto. Così rimasero pe  
r tutta la vita cieche, come punizione per la loro cattiveria.  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ head -1 cenerentola  
C'era una volta un uomo ricco, che rimase vedovo molto presto; egli aveva una fi  
glia, e ogni giorno andavano a trovare la mamma ed a portarle dei fiori. Dopo qu  
alche tempo il padre si risposò con una donna che aveva già due figlie, così ess  
i andarono a vivere nella casa dell'uomo. Le figlie della donna erano dispettose  
e monelle, così per la piccola bambina non furono giorni felici. Esse dicevano,  
"dovrebbe stare in salotto con noi? Chi mangia il pane deve guadagnarselo!" Le  
tolsero i suoi bei vestiti, le fecero indossare una vecchia palandrana grigia, e  
le diedero un paio di zoccoli. "Guardate la principessa, come è agghindata!" es  
clamarono ridendo e la condussero in cucina. Là dovette sgobbare da mattina a se  
ra, alzarsi prima di giorno, portare l'acqua, accendere il fuoco, cucinare e lav  
are. Per giunta le sorelle gliene facevano di tutti i colori, la schernivano e l  
e versavano ceci e lenticchie nella cenere, sicché doveva raccogliarli a uno a u  
no. La sera, dopo tante fatiche, non andava a letto, ma si coricava nella cenere  
accanto al focolare. E siccome era sempre sporca e impolverata, la chiamavano C  
enerentola.  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ clear
```

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Comando: ls

Il comando **ls** (**list directory**) in Linux è uno dei comandi più utilizzati per **visualizzare il contenuto di directory**. Esso elenca i file e le directory presenti nella **directory corrente** o in una **directory specificata** in ordine alfabetico.

\$ ls [opzioni] [directory]

Esempio:

\$ ls - mostra file/directory nella directory corrente

Esempio:

\$ ls home - mostra file/directory nella directory *home*

Esempio:

\$ ls --color=never - mostra senza l'uso colori

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ls  
cenerentola Immagini Musica prova2 prova4 Pubblici Scrivania Video  
Documenti Modelli prova1 prova3 prova5 Scaricati snap  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ls /home  
marco-cascio-its  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ls /  
bin home mnt sbin usr-is-merged usr  
bin.usr-is-merged lib opt snap var  
boot lib64 proc srv  
cdrom lib.usr-is-merged root swap.img  
dev lost+found run sys  
etc media sbin tmp  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ls --color=never  
cenerentola Immagini Musica prova2 prova4 Pubblici Scrivania Video  
Documenti Modelli prova1 prova3 prova5 Scaricati snap  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Comando: ls -l e file nascosti

Per ottenere **informazioni più dettagliate** sui file bisogna usare l'opzione **-l**

```
$ ls -l
```

In Linux esistono anche i **file nascosti** che iniziano con il carattere **.** :

- file di configurazione che i programmi collocano nelle home-directory degli utenti per memorizzare le impostazioni utente

Per **visualizzare** anche i **file nascosti** bisogna utilizzare l'opzione **-a**

```
$ ls -a
```

- mostra i file nascosti e non nascosti

```
$ ls -al
```

- combina le opzioni **-a** e l'opzione **-l**

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ls -al  
totale 96  
drwxr-x--- 15 marco-cascio-its marco-cascio-its 4096 giu 25 17:55 .  
drwxr-xr-x  3 root              root          4096 giu 20 13:47 ..  
-rw----- 1 marco-cascio-its marco-cascio-its  134 giu 25 17:31 .bash_history  
-rw-r--r-- 1 marco-cascio-its marco-cascio-its  220 mar 31 10:41 .bash_logout  
-rw-r--r-- 1 marco-cascio-its marco-cascio-its 3771 mar 31 10:41 .bashrc  
drwx----- 11 marco-cascio-its marco-cascio-its 4096 giu 20 13:50 .cache  
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its 11371 giu 25 17:55 cenerentola  
drwx----- 11 marco-cascio-its marco-cascio-its 4096 giu 20 13:56 .config  
drwxr-xr-x  2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Documenti  
drwxr-xr-x  2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Immagini  
-rw----- 1 marco-cascio-its marco-cascio-its   20 giu 24 16:23 .lessht  
drwx----- 4 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 .local  
drwxr-xr-x  2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Modelli  
drwxr-xr-x  2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Musica  
-rw-r--r-- 1 marco-cascio-its marco-cascio-its  807 mar 31 10:41 .profile  
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its   61 giu 25 17:26 prova1  
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its    0 giu 25 17:26 prova2  
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its    0 giu 25 17:26 prova3  
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its    0 giu 25 17:26 prova4  
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its    0 giu 25 17:26 prova5  
drwxr-xr-x  2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Pubblici  
drwxr-xr-x  2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Scaricati
```

Lettura dei file nascosti

I comandi:

```
$ cat .profile
```

```
$ cat .bash_history
```

consentono di leggere i file nascosti chiamati *profile* e *bash_history*.

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ cat .profile  
# ~/.profile: executed by the command interpreter for login shells.  
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login  
# exists.  
# see /usr/share/doc/bash/examples/startup-files for examples.  
# the files are located in the bash-doc package.  
  
# the default umask is set in /etc/profile; for setting the umask  
# for ssh logins, install and configure the libpam-umask package.  
#umask 022  
  
# if running bash  
if [ -n "$BASH_VERSION" ]; then  
    # include .bashrc if it exists  
    if [ -f "$HOME/.bashrc" ]; then  
        . "$HOME/.bashrc"  
    fi  
fi  
  
# set PATH so it includes user's private bin if it exists  
if [ -d "$HOME/bin" ] ; then  
    PATH="$HOME/bin:$PATH"  
fi
```

```
marco-cascio-its@marco-cascio-its-VirtualBox: ~  
marco-cascio-its@marco-cascio-its-VirtualBox:~$ cat .bash_history  
logout  
exit  
who  
login  
nano prova1  
touch prova1  
touch prova2  
touch prova3 prova4 prova5  
nano prova2  
nano prova1  
cat prova1  
logout  
exit  
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Altre opzioni per il comando ls

Esistono molte altre opzioni per il comando **ls**. Per esempio:

\$ **ls -lt**: mostra un elenco dettagliato dei file secondo la data dell'ultima modifica

\$ **ls -lS**: ordina i file dal più grande al più piccolo

\$ **ls -lr**: ordina i file dal più piccolo al più grande

Esempio: \$ **ls -al** etc - mostra tutti i file nascosti e non, contenuti nella directory *etc* in modo dettagliato

Esempio: \$ **ls -alS** etc - come sopra ma ordinandoli dal più grande al più piccolo

File System Information/1

Esaminiamo più in dettaglio le informazioni ricevute dal comando `ls -l`:

Esempio:

```
$ ls -l /home/marco-cascio
```

```
-rw- rw-r- -      1  marco-cascio marco-cascio 11371  giu  25   17:26  prova1
drwxr-xr-x        2  marco-cascio marco-cascio 4096   giu  20   13:47  Documents
lrwxrwxrwx        1  marco-cascio marco-cascio  19     nov  29   13:22  prova1→/home/renata/prova1
```

Il primo carattere è:

- - se si tratta di un **file**
- **d** se si tratta di un **directory**
- **l** se si tratta di un **link simbolico** (puntatore)

File System Information/2

User (u)			Group (g)			Others (o)		
r	w	x	r	w	x	r	w	x

```
-rw- rw- r--      1  marco-cascio marco-cascio 11371  giu 25   17:26 prova1
drwx r-x r-x      2  marco-cascio marco-cascio 4096   giu 20   13:47 Documents
lrwx rwx rwx      1  marco-cascio marco-cascio 19      nov 29   13:22 prova1→/home/renata/prova1
```

I successivi gruppi di tre caratteri rappresentano i **permessi** attribuiti al file rispettivamente per il **proprietario del file**, per il **gruppo** e per il **resto del mondo**. Essi possono assumere i valori:

- **r** permesso di lettura
- **w** permesso di scrittura
- **x** permesso di esecuzione (per le directory rappresenta il permesso di attraversamento)
- **-** nessun permesso

File System Information/3

-rw- rw- r--	1	marco-cascio	marco-cascio	11371	giu 25	17:26	prova1
drwx r-x r-x	2	marco-cascio	marco-cascio	4096	giu 20	13:47	Documents
lrwx rwx rwx	1	marco-cascio	marco-cascio	19	nov 29	13:22	prova1→/home/renata/prova1

- Il primo carattere della 1° colonna indica se l'elemento è un **file**, una **directory** o un **link simbolico**
- Il successivi caratteri della 1° colonna indicano i **permessi**
- la 2° colonna è il numero di collegamenti fisici (**hard link**) al file
- la 3° e la 4° colonna indicano l'**utente** e il **gruppo proprietario** del file
- la 5° colonna rappresenta la dimensione in byte del file
- la 6° e la 7° colonna rappresentano la data e l'ora dell'ultima modifica del file
- l'ultima colonna è il nome del file

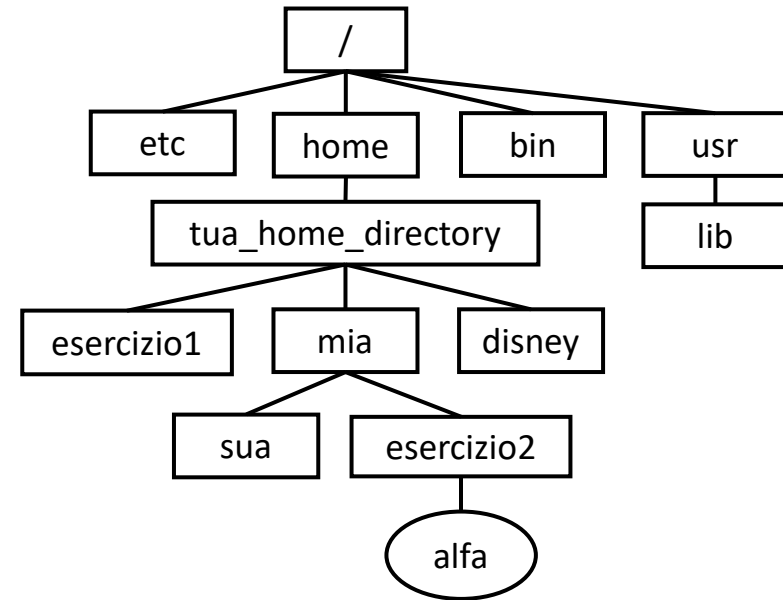
N.B. in caso di collegamenti simbolici (link simbolico), '→' è la destinazione del collegamento

Creazione di una directory/1

Per creare una nuova directory il comando è:

```
$ mkdir [nome_nuova_directory]
```

Il comando crea la directory *nome_nuova_directory* all'interno della directory corrente, ma è possibile specificare la path desiderata.



Creazione di una directory/2

Esempio:

\$ mkdir *mia*

- crea una directory chiamata *mia* nella directory corrente

Esempio:

\$ mkdir *new*

- crea una directory chiamata *new* nella directory corrente

Esempio:

\$ mkdir *tua disney prova*

- crea tre directory chiamate *tua*, *disney* e *prova* nella directory corrente

Esempio:

\$ mkdir *mia/sua*

- crea la directory *sua* all'interno della directory *mia*

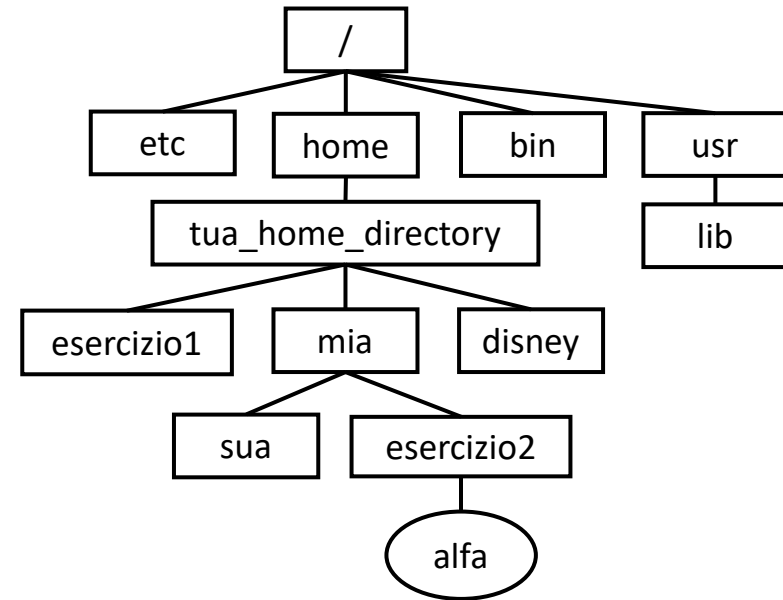
Cambiare directory/1

Il comando per **cambiare directory** è:

\$ cd *[directory]*

La directory specificata diviene la **working directory**.

Se nessuna directory viene specificata, si ritorna alla home directory.



Cambiare directory/2

Esempio:

\$ **cd** *mia*

- ora la directory corrente è */home/marco-cascio/mia*

\$ **pwd**

- verifica la directory corrente)

Esempio:

\$ **cd** *esercizio2*

- ora la directory corrente è */home/marco-cascio/mia/esercizio2*

\$ **pwd**

- verifica la directory corrente

Esempio:

\$ **cd**

- si ritorna alla home directory */home/marco-cascio*

\$ **pwd**

- verifica la directory corrente

Spostamenti assoluti

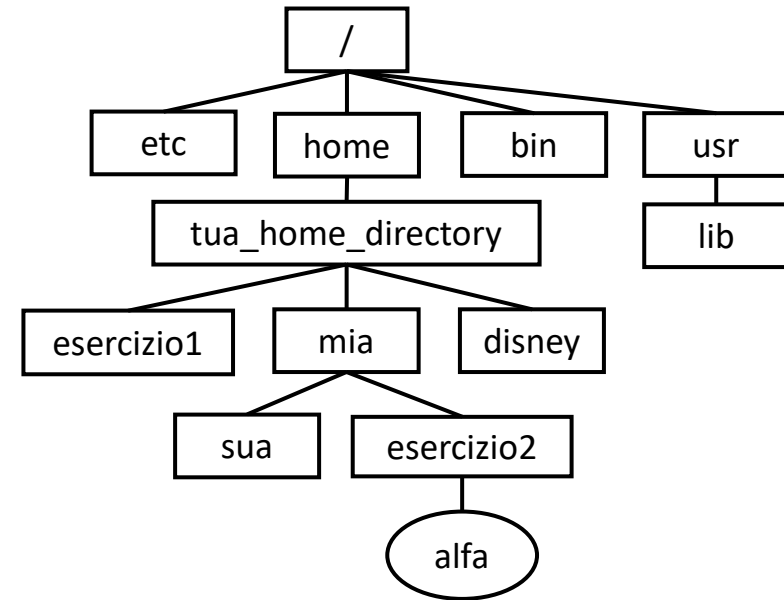
Il **percorso assoluto** parte dalla **directory radice**: /

```
$ cd /dir1/dir2
```

La directory */dir1/dir2* diviene la **working directory**.

Esempio:

```
$ cd /usr
```

 - spostamento assoluto nella directory */usr*

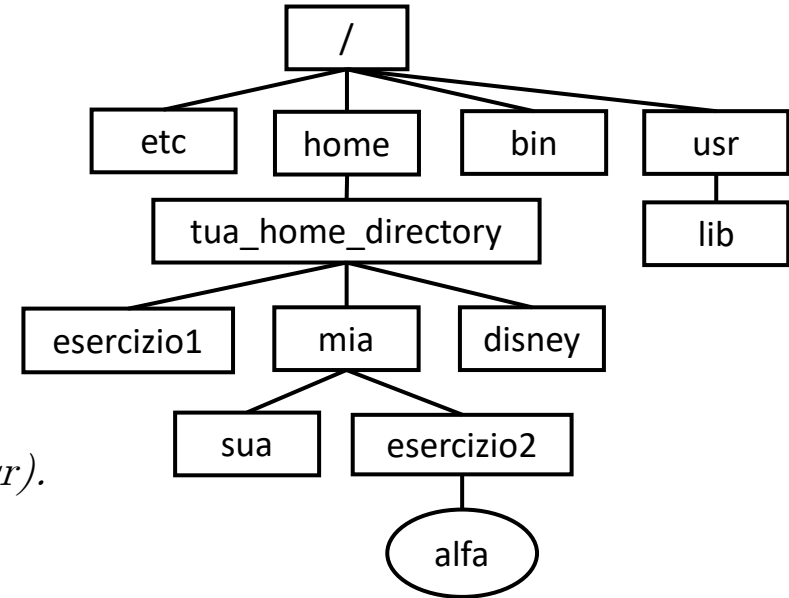
Spostamenti relativi

Il **percorso relativo** parte dalla **posizione corrente**

`$ cd dir1`

La directory *dir1* diviene la **working directory**.

`$ cd lib` - *lib* diventa la working directory (partendo da */usr*).



Spostamento a ritroso

Il riferimento alla **directory corrente** può essere rappresentato da un **.** (**punto**)

\$ cd ./disney - dalla directory corrente */home/marco-cascio*, spostati nella directory chiamata *disney*

Per spostarsi dalla **directory corrente** a quella **genitore** si usano **..** (**due punti**)

\$ cd .. - dalla directory *disney* ritorna alla directory */home/marco-cascio*

Directory di Sistema – Percorsi assoluti/1

- \$ `cd /` - directory radice del sistema
- \$ `cd /bin` - contiene i programmi di sopravvivenza per il sistema

- \$ `cd /boot` - contiene i dati necessari all'avvio del sistema
- \$ `cd /dev` - contiene i cosiddetti devices
- \$ `cd /etc` - contiene i file di configurazione
- \$ `cd /lib` - contiene le librerie, ovvero parti di codice condivise tra i programmi

- \$ `cd /mnt` - contiene i mount-point
- \$ `cd /proc` - il mount-point del file system proc
- \$ `cd /root` - è la home directory dell'amministratore di sistema, utente *root*
- \$ `cd /sbin` - contiene programmi che solitamente richiedono i privilegi di *root*

Directory di Sistema – Percorsi assoluti/2

\$ `cd /tmp` - contiene i file temporanei

\$ `cd /var` - contiene alcuni file in attesa di essere processati, come file di stampa

Verifica del tipo di file

Per **verificare** di che **tipo** sia un **file** (eseguibile, dati, directory) si può utilizzare il comando **file**.

Esempio: **\$ file prova1.txt**

Output: *prova1.txt: Unicode text, UTF-8 text*

Esempio: **\$ file prova2.txt**

Output: *prova2.txt: empty*

Esempio: **\$ file /home**

Output: *home: directory*

Esempio **\$ file /bin/ls**

Output: */bin/ls: ELF 64-bit LSB pie executable*

Permessi sui file espressi in forma di stringa/1

Ad ogni **file** o **directory** in un sistema Linux sono associati **permessi** per:

- possessore del file stesso (**u**);
- membri del suo gruppo (**g**) ;
- altri utenti (**o**);

I permessi possono essere di:

- lettura (**r**)
- scrittura (**w**)
- esecuzione o attraversamento per le directory (**x**)
- nessun permesso (-)

User (u)			Group (g)			Others (o)		
r	w	x	r	w	x	r	w	x

Permessi sui file espressi in forma di stringa/2

- **Read (r):**
 - **file:** possibilità di leggere il contenuto
 - **directory:** leggere l'elenco dei file contenuti in una directory
- **Write (w):**
 - **file:** possibilità di modificare il contenuto
 - **directory:** possibilità di aggiungere, rimuovere, rinominare file
- **Execute (x):**
 - **file:** possibilità di eseguire il file
 - **directory:** possibilità di attraversare una directory o accedervi tramite nome del percorso

Permessi sui file espressi in forma di stringa/3

Il comando per vedere i permessi associati ad un file è `ls -l`

Esempio: `$ ls -l prova1.txt`

Output: `-rw- rw- r-- 1 marco-cascio marco-cascio 11371 giu 25 17:26 prova1.txt`

Questo file ha il permesso di:

- **Lettura (r) e scrittura (w)** per l'utente, il quale non ha il permesso di esecuzione (-)
- **Lettura (r) e scrittura (w)** per il gruppo di utenti presenti sul sistema, i quali non hanno il permesso di esecuzione (-)
- **di sola lettura (r)** per il resto del mondo, per cui non è possibile la scrittura (-) e l'esecuzione (-) di tale file

Definire i permessi

Il comando che si usa per stabilire i permessi è:

```
$ chmod [ugoa] [+ - =] [r w x] [nome_file]
```

dove *[ugoa]* indicano:

- u: proprietario (user)
- g: gruppo (group)
- o: altri utenti (other)
- a: tutti (all)

dove *[+ - =]* indicano:

- + aggiungi
- - rimuovi
- = assegna

dove *[r w x]* indicano:

- r: read
- w: write
- e: execute

Definire i permessi – Esempio 1

Esempio 1

Creare un file *prova.txt* ed assegnare al file i seguenti permessi:

- **user:** permesso di lettura, scrittura, esecuzione
- **gruppo:** permesso di lettura e scrittura
- **altri:** nessun permesso

Soluzione:

```
$ touch prova.txt
```

```
$ chmod u=rwx,g=rw,o-rwx prova.txt
```

```
$ ls -l prova.txt
```

Output: *-rwx rw- ---*

Definire i permessi – Esempio 2

Esempio 2

Creare un file *prova.txt* e proteggere totalmente il file dal gruppo e dagli altri. Ciò significa che dobbiamo negare tutti i permessi al gruppo e agli altri, mentre lasciare i permessi di lettura, scrittura ed esecuzione al solo utente.

Soluzione:

```
$ touch prova.txt
```

```
$ chmod u=rwx,go-rwx prova.txt
```

```
$ ls -l prova.txt
```

```
Output: -rwx --- ---
```

Definire i permessi – Esempio 3

Esempio 3

Creare un file *prova.txt* e assegnare al file i seguenti permessi:

- **user:** lettura, esecuzione
- **gruppo:** lettura
- **altri:** nessun permesso

Soluzione:

```
$ touch prova.txt
```

```
$ chmod u=rx,g=r,o-rwx prova.txt
```

```
$ ls -l prova.txt
```

```
Output: -r-x r-- ---
```

Copia di un file

Per effettuare la **copia** di un **file** il comando è **cp** (**copy**).

\$ cp [file da copiare] [file copia] [nome percorso della copia - opzionale]

Esempio:

\$ cp pluto.txt pluto2.txt - crea il file *pluto2.txt* che è una copia del file *pluto.txt*

Esempio:

\$ cp prova1.txt pluto3.txt - crea il file *pluto3.txt* che è una copia del file *prova1.txt*

Copia di file in una directory

Esempio:

\$ **cp** *prova.txt prova3.txt mia* - crea una copia del file *prova.txt*, *prova3.txt* all'interno della directory *mia*

Esempio:

\$ **cp** * *mia/sua* - copia tutti i file nella directory corrente dentro la directory *mia/sua*

\$ **cp** *pl** *mia/sua* - copia tutti i file il cui nome inizia per *pl* nella directory *mia/sua*

Copia di una directory/1

Per effettuare la copia di una directory incluso il suo contenuto la sintassi è:

```
$ cp -r [dir1] [dir2]
```

Esempio (se *dir2* non esiste):

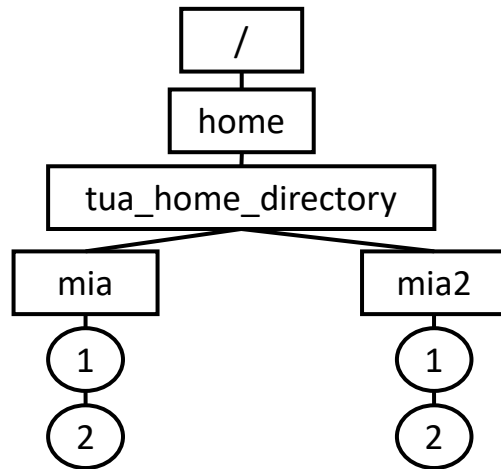
```
$ cp -r mia mia2           - copia solo il contenuto della directory mia in una nuova directory mia2
```

Esempio (se *dir2* esiste):

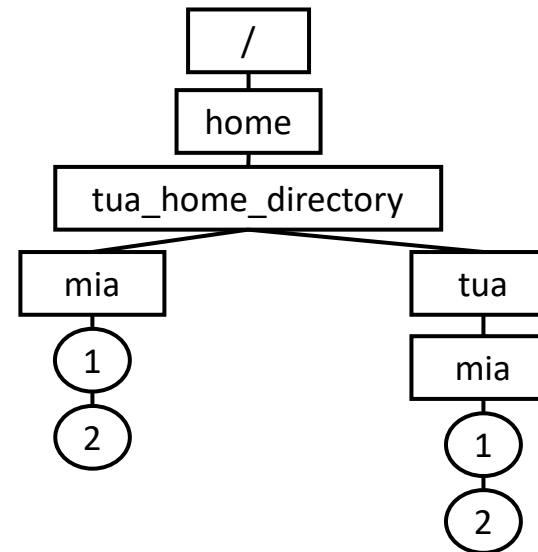
```
$ cp -r mia tua           - copia la directory mia e il suo contenuto dentro la directory tua
```

Copia di una directory/2

\$ `cp -r mia mia2`



\$ `cp -r mia tua`



Eliminare un file o una directory/1

Per **eliminare** un **file** o una **directory** il comando è **rm** (**remove**).

\$ rm [file] - nel caso in cui si vuole eliminare un file

\$ rm -r [dir] - nel caso in cui si vuole eliminare una directory e il suo contenuto

\$ rmdir [dir] - nel caso in cui si vuole eliminare una directory vuota

Eliminare un file o una directory/2

Esempio:

\$ **rm** *prova.txt* - elimina il file *prova.txt*

Esempio:

\$ **rm** * - elimina tutti i file nella directory corrente

\$ **rm** *pl** - elimina tutti i file il cui nome inizia per *pl*

Esempio:

\$ **rmdir** *new* - elimina la directory *new* (solo se vuota)

Esempio:

\$ **rm -r** *mia2* - elimina la directory *mia2* con tutto il suo contenuto

Eliminare un file o una directory/3

Con il comando

\$ rm -ri [dir]

si elimina la directory indicata e il suo contenuto chiedendo le conferme.

Esempio:

\$ rm -ri tua

```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ rm -ri tua
rm: entrare nella directory 'tua'? si
rm: entrare nella directory 'tua/mia'? si
rm: rimuovere file regolare 'tua/mia/pluto'? si
rm: entrare nella directory 'tua/mia/esercizio2'? si
rm: rimuovere file regolare vuoto 'tua/mia/esercizio2/alfa'? si
rm: rimuovere directory 'tua/mia/esercizio2'? si
rm: entrare nella directory 'tua/mia/sua'? si
rm: rimuovere file regolare 'tua/mia/sua/pluto'? si
rm: rimuovere file regolare 'tua/mia/sua/lezione'? si
rm: rimuovere file regolare 'tua/mia/sua/pluto3'? si
rm: rimuovere file regolare 'tua/mia/sua/pluto2'? si
rm: rimuovere directory 'tua/mia/sua'? si
rm: rimuovere file regolare 'tua/mia/pluto3'? si
rm: rimuovere directory 'tua/mia'? si
rm: rimuovere directory 'tua'? si
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Eliminare un file o una directory/4

ATTENZIONE!!!!

Da utente privilegiato non eseguire MAI i comandi:

`$ rm -rf .*`

- cancella tutti i file e le directory nella working directory

`$ rm -rf /`

- cancellerà tutti i file e le directory nel file system a partire dalla root /

`$ rm -rf --no-preserve-root /`

- come sopra ma bypassando meccanismi di protezione

Spostare e/o rinominare un file/1

Il comando per **rinominare** o **spostare** un **file** o una **directory** è **mv** (move).

- `$ mv [file1] [file2]` - rinomina il *file1* in *file2*
- `$ mv [file1] [dir1/dir2]` - sposta *file1* dalla directory corrente alla directory *dir1/dir2*
- `$ mv [file1] [dir1/dir2/file2]` - sposta *file1* nella directory *dir1/dir2* cambiandogli nome in *file2*

Spostare e/o rinominare un file/2

Esempio:

\$ **mv** *prova.txt helloWorld.txt* - rinomina il file *prova.txt* in *helloWorld.txt*

Esempio:

\$ **mv** *cenerentola.txt ./disney* - sposta il file *cenerentola.txt* nella directory *disney*

Esempio:

\$ **mv** *cenerentola.txt ./disney/brontolo.txt* - sposta il file *cenerentola.txt* nella directory *disney* rinominandolo in *brontolo.txt*

Comando: grep

Il comando **grep** è utilizzato per la **ricerca di stringhe** all'interno di un **file**.

Esempio: **\$ grep volta ./disney/cenerentola.txt**

(ricerca la stringa *volta* all'interno del file *cenerentola.txt*, contenuto nella directory */home/marco-cascio/disney*)

- Per ricercare espressioni contenenti **SPAZI** oppure **TAB** si possono usare apici.

Esempio: **\$ grep 'uomo ricco' ./disney/cenerentola.txt**

(ricerca la stringa *'uomo ricco'* all'interno del file *cenerentola*)

- L'opzione **-i** serve ad ignorare se i caratteri sono maiuscoli o minuscoli.

Esempio: **\$ grep -i pettinaci ./disney/cenerentola.txt**

(ricerca la stringa *pettinaci* all'interno del file *cenerentola* in modo *no case sensitive*)

Comando: find/1

Il comando **find** serve ad effettuare la **ricerca** di un **file**.

```
$ find [nomedir] [maxdepth] -name [nomefile]
```

dove

- *[nomedir]* è il percorso della directory da cui effettuare la ricerca
- *[maxdepth]* è il numero di livelli di directory in cui cercare
- *[nomefile]* è il nome del file da cercare

Comando: find/2

Esempio: `$ find . -name pluto`

(cerca il file *pluto* nella working directory e sub-directory)

Esempio: `$ find . -name pluto -exec cat {} \;`

(cerca il file *pluto* ed esegue il comando *cat*. `\;` chiuda la sequenza di comandi da eseguire)

```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ find . -name pluto
./mia/pluto
./mia/sua/pluto
marco-cascio-its@marco-cascio-its-VirtualBox:~$ find . -name pluto -exec cat {}
\;
Ciao, sono Pluto! Bau-Bau!
Ciao, sono Pluto! Bau-Bau!
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```


Comando: find/3

Esempio: `$ find . -maxdepth 1 -name pluto`

(cerca il file pluto solo nella working directory)

Esempio: `$ find . -maxdepth 2 -name pluto`

(cerca il file pluto nella working directory e nelle sub-directory di primo livello)

Esempio: `$ find . -empty`

(cerca i file vuoti nella working directory e sub-directory)

Comando: tree

Il comando **tree** serve ad **esplorare** in **maniera grafica**, ad **albero**, una directory specificata.

Se il comando non viene trovato è possibile installarlo mediante: **sudo apt install tree**

- \$ **tree** - stampa l'albero della working directory
- \$ **tree [dir]** - stampa l'albero della directory dir specificata
- \$ **tree -d** - mostra solo le directory senza mostrare i file

Risultato \$ **tree**

```
disney
├── brontolo
├── cenerentola
├── Documenti
├── helloWorld
├── Immagini
├── mia
│   ├── esercizio2
│   │   └── alfa
│   ├── pluto
│   ├── pluto3
│   └── prova
│       ├── prova2
│       ├── prova3
│       ├── prova4
│       └── prova5
```

Risultato \$ **tree -d**

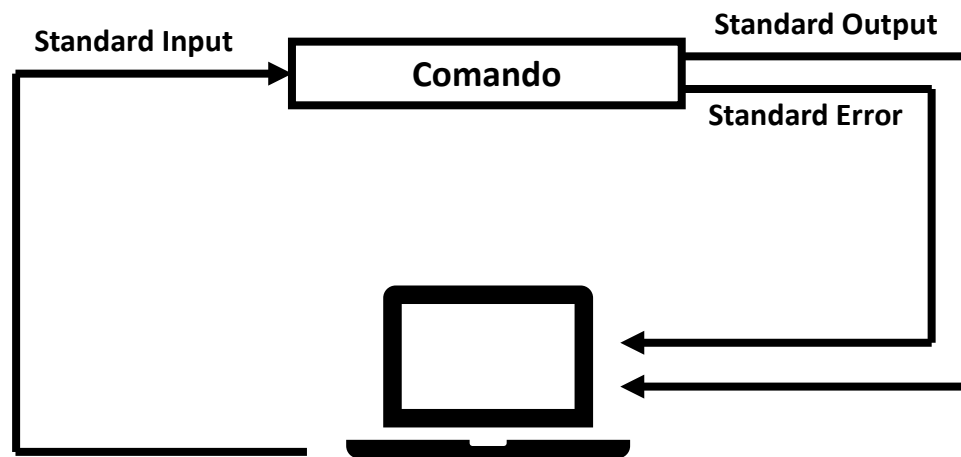
```
disney
Documenti
Immagini
mia
├── esercizio2
├── prova
├── sua
Modelli
Musica
Pubblici
Scaricati
Scrivania
snap
```

Redirezione dell' Input/Output

Quando un **comando** viene **eseguito** ha alcuni **canali standard** per il **flusso dei dati**:

- **standard input**: dati di ingresso
- **standard output**: dati di uscita
- **standard error**: eventuali messaggi di errore

La **redirezione** è la **deviazione** dei canali standard di un dato **comando** verso **destinazioni** (o da **sorgenti**, nel caso dello standard input) che sono **diverse** da quelle **predefinite**.



Standard Input

Per **redirezionare** lo **standard input** si utilizza il simbolo: `<`

Il **comando** che **precede** il simbolo `<` considera come **input** il **contenuto del file** specificato subito **dopo**.

Esempio:

- Inserire nella directory `/home/[tua_home_directory]` i file `sum.sh` e `number.txt` forniti con le slide
- Definire i permessi di esecuzione per il file `sum.sh`: `$ chmod u+x sum.sh`
- Scrivere: `$. /sum.sh < number.txt`
- Lo script `sum.sh` viene eseguito prendendo in **input** il numero (10) contenuto nel file `number.txt`

```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ./sum.sh < number.txt
10 + 10 = 20
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

Standard Output/1

Per **redirezionare** lo **standard output** si utilizzano i simboli: `>` oppure `>>`

L'**output** del **comando** che **precede** il simbolo `>` viene **rediretto** nel **file specificato** subito **dopo**, creandolo o sovrascrivendolo.

Esempio: `$ ls > folder_list.txt`

(l'output del comando **ls** viene scritto nel nuovo file *folder_list.txt*)

Esempio: `$ echo 20 > number.txt`

(l'output del comando **echo** è stato scritto nel file esistente *number.txt* che viene sovrascritto)

```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo "20" > number.txt
marco-cascio-its@marco-cascio-its-VirtualBox:~$ cat number.txt
20
```

Standard Output/2

L'**output** del **comando** che **precede** la sequenza `>>` viene aggiunto al **file specificato**.

Nel caso in cui questo **file non esista**, questo viene **creato** e, quindi, riempito con l'output del comando.

Esempio: `$ echo 30 >> number.txt`

(l'output del comando **echo** è stato aggiunto al file *number.txt*)

```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo "30" >> number.txt
marco-cascio-its@marco-cascio-its-VirtualBox:~$ cat number.txt
20
30
```

Comando: echo

Il comando **echo** viene utilizzato per **visualizzare** una **linea di testo** o una **variabile** nel **terminale**.

Esempio: **\$ echo Ciao Mondo!**

Output: *Ciao Mondo!* <- stampa su terminale

Esempio **\$ echo Ciao Mondo! > mio_file.txt**

Output: *Ciao Mondo!* <- scritto all'interno di *mio_file.txt* sovrascrivendolo

Esempio: **\$ echo mi aggiungo al file! >> mio_file.txt**

Output: *Ciao Mondo!*

mi aggiungo al file! <- aggiunto all'interno di *mio_file.txt*

Pipeline

La **pipeline** è una forma di **redirezione** in cui la shell **invia l'output** di un **comando** come **input** del **comando successivo** utilizzando il simbolo **barra verticale**: |

command1 | *command2* | *command3* | ...

L'output del *command1* diventa l'input per il *command2*.

L'output del *command2* diventa l'input del *command3* e così via.



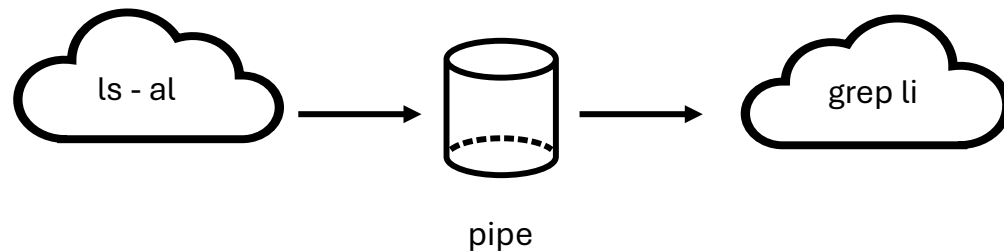
Pipeline – Esempio 1

Esempio 1

Scrivere, utilizzando la pipe, il comando per visualizzare da un elenco esteso della directory */home* tutti i file e le directory che nel loro nome contengono le lettere *li*.

Soluzione:

```
$ ls -al | grep li
```



```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ ls -al | grep li
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its 145 lug  2 21:16 folder_list
drwxr-xr-x 2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Modellli
-rw-rw-r-- 1 marco-cascio-its marco-cascio-its 139 lug  2 20:20 multiplica
drwxr-xr-x 2 marco-cascio-its marco-cascio-its 4096 giu 20 13:47 Pubblici
marco-cascio-its@marco-cascio-its-VirtualBox:~$
```

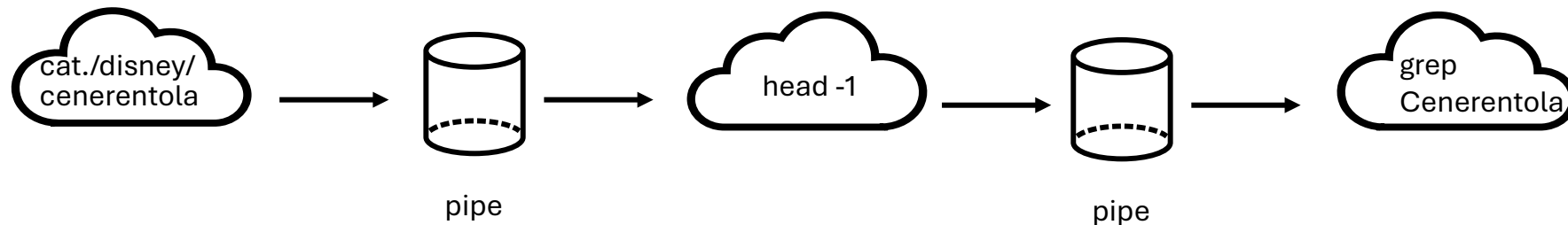
Pipeline – Esempio 2

Esempio 2

Scrivere, utilizzando la pipe, il comando per cercare la parola *Cenerentola* nella prima riga del file *cenerentola.txt*.

Soluzione:

```
$ cat ./disney/cenerentola.txt | head -1 | grep Cenerentola
```



```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ cat ./disney/cenerentola | head -1 | grep Cenerentola
```

```
C'era una volta un uomo ricco, che rimase vedovo molto presto; egli aveva una figlia, e ogni giorno andavano a trovare l  
a mamma ed a portarle dei fiori. Dopo qualche tempo il padre si risposò con una donna che aveva già due figlie, così ess  
i andarono a vivere nella casa dell'uomo. Le figlie della donna erano dispettose e monelle, così per la piccola bambina  
non furono giorni felici. Esse dicevano, "dovrebbe stare in salotto con noi? Chi mangia il pane deve guadagnarselo!" Le  
tolsero i suoi bei vestiti, le fecero indossare una vecchia palandrana grigia, e le diedero un paio di zoccoli. "Guardat  
e la principessa, come è agghindata!" esclamarono ridendo e la condussero in cucina. Là dovette sgobbare da mattina a se  
ra, alzarsi prima di giorno, portare l'acqua, accendere il fuoco, cucinare e lavare. Per giunta le sorelle gliene faceva  
no di tutti i colori, la schernivano e le versavano ceci e lenticchie nella cenere, sicché doveva raccogliarli a uno a u  
no. La sera, dopo tante fatiche, non andava a letto, ma si coricava nella cenere accanto al focolare. E siccome era semp  
re sporca e impolverata, la chiamavano Cenerentola.
```

Lista/1

Una **lista di comandi** è una sequenza costituita da **uno o più comandi** separati da: ;
Questi comandi sono **eseguiti** nell'**ordine** in cui appaiono nella sequenza.

command1 ; command2 ; command3 ; ...

Esempio: \$ **echo** *Buongiorno!* ; **echo** *La data di oggi!* ; **date** ; **cal**

(se il comando **cal** non viene trovato è possibile installarlo mediante **sudo apt install ncal**)

```
marco-cascio-its@marco-cascio-its-VirtualBox:~$ echo Buongiorno! ; echo La data di oggi! ; date ; cal
Buongiorno!
La data di oggi!
mer 3 lug 2024, 11:44:19, CEST
    Luglio 2024
do lu ma me gi ve sa
   1  2  3  4  5  6
  7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Lista/2

Le parentesi tonde (***command1 ; command2 ; command3***) nella shell vengono utilizzate per raggruppare comandi insieme in modo che vengano eseguiti in una sotto-shell. Una sotto-shell è un ambiente separato che esegue i comandi come se fossero un singolo comando.

Esempio: `$ echo Ciao ; echo Caro! > out_file.txt`

(esegue entrambi i comandi, ma *Ciao* viene visualizzato su terminale, mentre l'output dell'ultimo **echo Caro!** viene scritto in un file *out_file.txt*)

Esempio: `$ (echo Ciao ; echo "Caro!") > out_file.txt`

(esegue entrambi i comandi e scrive l'output complessivo, formato dalle parole *Ciao e Caro!*, nel file *out_file.txt*)

Operatore logico: && (AND)

L'operatore **&&** esegue il **comando successivo** solo se il **comando precedente** è stato eseguito con **successo**.

command1 && command2 (*command2* viene eseguito solo se *command1* è stato eseguito con successo)

Esempio (se *out_file.txt* esiste): **\$ rm out_file.txt && echo il file è stato eliminato!**

Output: *il file è stato eliminato!* (perchè **\$ rm out_file.txt** è stato eseguito con successo)

Esempio (se *out_file.txt* non esiste): **\$ rm out_file.txt && echo il file è stato eliminato!**

Output: *rm: impossibile rimuovere file out_file.txt ...* (non è stato possibile eseguire il comando **echo**)

Operatore logico: || (OR)

L'operatore `||` esegue il **comando successivo** solo se l'esecuzione del **comando precedente** è **fallita**.

`command1 || command2` (`command2` viene eseguito solo se `command1` è stato eseguito senza successo)

Esempio (il file `out_text.txt` non esiste): `$ rm out_file.txt || echo il file non esiste!`

Output: *il file non esiste!* (perché non è stato possibile eseguire il comando `$ rm out_file.txt`)