

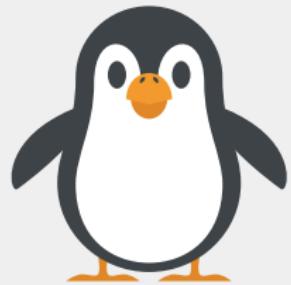
SHELL LINUX & BASH SCRIPTING

TUTORATO SISTEMI OPERATIVI

DAVIDE CARNEMOLLA

DIPARTIMENTO DI MATEMATICA E INFORMATICA
UNIVERSITÀ DI CATANIA

2022/2023





Davide Carnemolla
@Herbrant



Open Source
Enthusiast



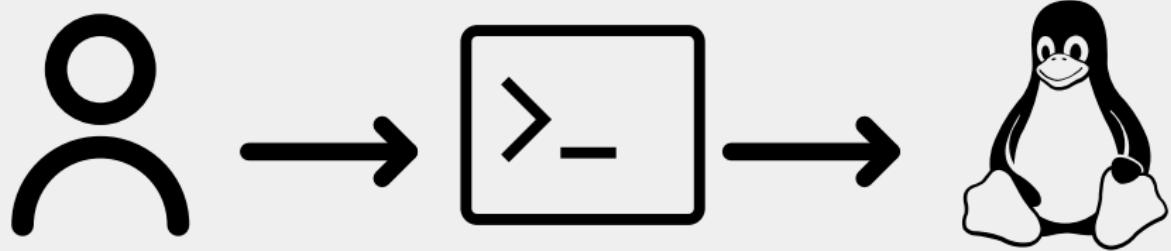
ExamBox's father



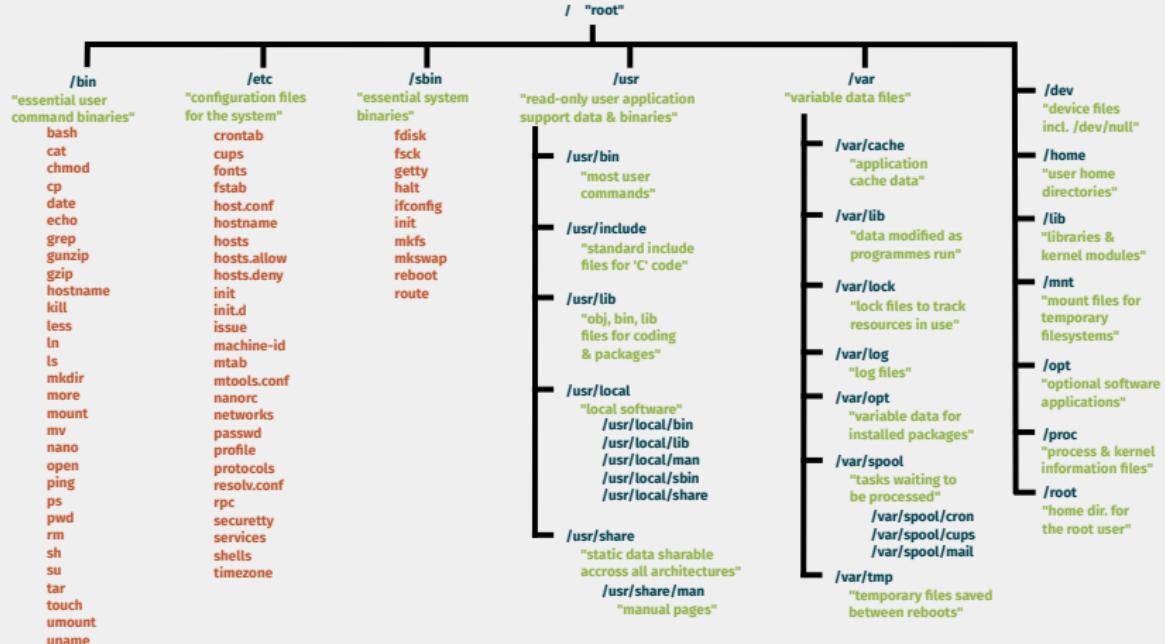
Ricotta lover

SHELL LINUX

LA SHELL



IL FILESYSTEM





Path assoluto

Il **percorso assoluto** di un file è il percorso che va dalla radice del filesystem allo stesso.

Path relativo

Fissando logicamente una particolare directory (in genere quella corrente) è possibile identificare un file attraverso il **percorso relativo** che va da tale directory ad esso.

I PATH

Nota

Il percorso assoluto di un file è il suo percorso relativo rispetto alla root.

Esempio

Directory corrente: /home/

Path assoluto: /home/davide/file.txt

Path relativo: davide/file.txt

Cartelle virtuali

- la cartella . indica la cartella stessa
- la cartella .. indica la cartella genitore (utile per navigare il filesystem)

PATH: FILE DI DISPOSITIVO

In ambiente UNIX esistono dei file speciali chiamati file di dispositivo: identificano una particolare periferica del sistema ed attraverso questo è possibile interagire con tale periferica.



A caratteri



A blocchi

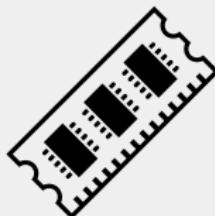
Esempi

- /dev/sda, /dev/sdb...
- /dev/sda1, /dev/sda2...
- /dev/tty1, /dev/tty2...

PATH: PSEUDO-DEVICES



`/dev/random`



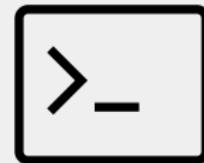
`/dev/shm`



`/dev/null`

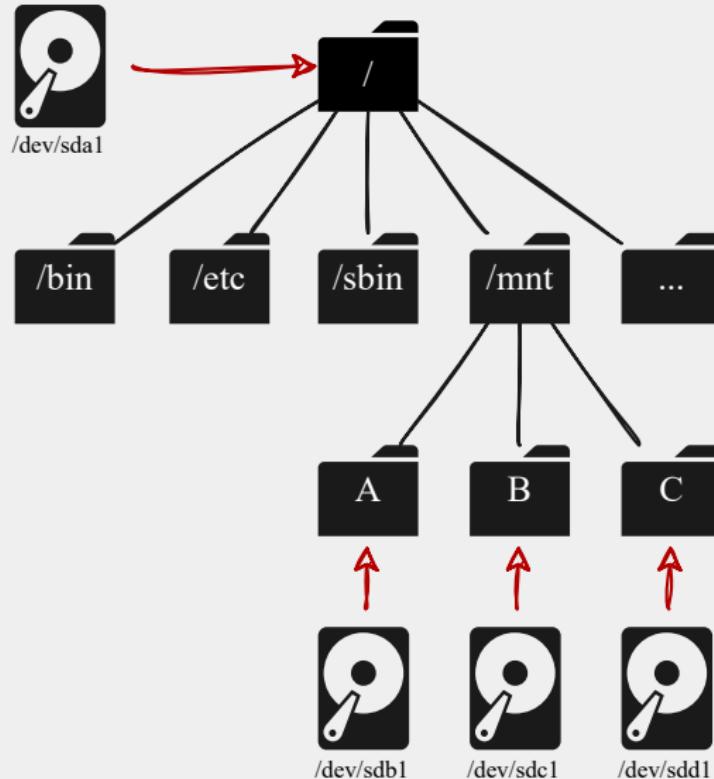


`/dev/full`



`/dev/ttyX`

MOUNT: QUANDO VOGLIO PIÙ SPAZIO



HELLO SHELL

```
$ echo "Hello shell"
```

- **echo** rappresenta il comando
- "**Hello world**" rappresenta un parametro per il comando

Parametri

Tipicamente un comando richiede dei parametri



File

123

Dati

• • •

Opzioni

CTRL + C CTRL + S CTRL + Q

“No maria, io esco”

*“Finisco di mangiare
la peperonata e
scendo! ”*

*“Ok ho digerito:
possiamo
riprendere.”*

COMANDI UTILI



"Non riesco a fare copia e incolla"

CTRL+SHIFT+C

CTRL+SHIFT+V

HELLO MAN



apropos stringa



whatis stringa



man stringa

WARNING



La sintassi dei comandi presentati è stata
semplificata per facilitare la presentazione

Per maggiori informazioni consultare sempre il manuale (**man**)

FILESYSTEM: LS

ls

Sintassi: **ls [-l] [-a] [-R] [pathname...]**

- **-l**: visualizza informazioni dettagliate
- **-a**: visualizza anche i file nascosti
- **-R**: visualizza il contenuto delle cartelle ricorsivamente
- **pathname**: oggetto del filesystem sul quale visualizzare le informazioni

Esempio

```
$ ls /home/davide/script.sh  
-rwxr--r-- 1 davide davide 967 31 mag 23.05 script.sh
```

PERMESSI DI ACCESSO

Nei sistemi Unix ogni file possiede dei permessi di accesso.
In principali sono:

- **r**: lettura
- **w**: scrittura
- **x**: esecuzione/attraversamento (directory)

Inoltre, il primo carattere è destinato a dei flag speciali:

- **d**: si tratta di una directory
- **l**: si tratta di un soft link
- **s**: attribuisce al file in esecuzione i privilegi dell'utente cui appartiene (SUID)

Esempio

```
-rwxr--r-- 1 davide davide 967 31 mag 23.05 script.sh
```

I METACARATTERI

Per abbreviare il nome di un file da specificare o per specificarne più di uno si possono utilizzare i metacaratteri:

- *: rappresenta una qualunque stringa di 0 o più caratteri
- ?: rappresenta un qualsiasi carattere
- []: singolo carattere tra quelli elencati
- {}: singola stringa tra quelle elencate

Esempio

```
$ ls  
esempio2.txt esempio3.txt esempio.txt script.sh scheda.pdf
```

```
$ ls esempio*.txt  
esempio2.txt esempio3.txt esempio.txt
```

```
$ ls esempio?.txt  
esempio2.txt esempio3.txt esempio.txt
```

```
$ ls *.{txt, pdf}  
esempio2.txt esempio3.txt esempio.txt scheda.pdf
```

FILESYSTEM: CD / PWD

cd

Sintassi: **cd [pathname]**

- **pathname:** nuova directory corrente (assoluto o relativo)

Il comando ci permette di navigare all'interno del filesystem cambiando di volta in volta la directory corrente.

pwd

Sintassi: **pwd**

Il comando ci permette di ottenere la *present working directory*, ovvero il path assoluto della directory in cui ci troviamo.

FILESYSTEM: MKDIR/RMDIR

mkdir

Sintassi: **mkdir [-p] pathname...**

- **-p**: non restituisce errori se la directory esiste già e crea tutte le directory necessarie per ottenere il path completo
- **pathname**: pathname da creare

Il comando **mkdir** crea le directory specificate mediante i parametri.

rmdir

Sintassi: **rmdir [-p] pathname...**

- **-p**: rimuove tutte le directory presenti nel pathname
- **pathname**: directory da eliminare

Il comando **rmdir** rimuove una directory vuota dal filesystem.

FILESYSTEM: CP



cp

Sintassi: **cp [-R] [-i] source... dest**

- **-R**: copia ricorsivamente il contenuto di una directory
- **-i**: chiede conferma prima di sovrascrivere i file
- **source**: oggetti da copiare
- **dest**: destinazione in cui copiare gli oggetti

Il comando **cp** copia i file specificati con **source** in **dest**.

FILESYSTEM: RM



rm

Sintassi: **rm [-r] [-i] [-f] pathname...**

- **-r**: elimina ricorsivamente tutti i file
- **-i**: chiede conferma prima di cancellare ogni file
- **-f**: cancella gli oggetti senza chiedere conferma
- **pathname**: oggetti da eliminare

Il comando **rm** elimina tutti i file specificati con i parametri.

rm -rf /

FILESYSTEM: MV



mv

Sintassi: **mv source... dest**

- **source:** file o directory da spostare
- **dest:** file o directory di destinazione

Il comando **mv** sposta il file o directory sorgente nella destinazione specificata.

FILESYSTEM: MOUNT

mount

Sintassi: **mount [-t fstype] [-o options] device mountpoint**

- **-t fstype**: permette di specificare il tipo del filesystem da montare (ext2, ext3, ext4, btrfs...)
- **-o options**: permette di specificare ulteriori opzioni per il mount
- **device**: il dispositivo da montare (di solito una partizione del tipo /dev/sdAX)
- **mountpoint**: directory in cui montare filesystem

Il comando **mount** permette di montare un filesystem all'interno dell'albero principale.

UTENTI NEI SISTEMI UNIX



root



sudo



standard user

```
robm@homebox ~$ sudo su  
Password:  
robm is not in the sudoers file.  
This incident will be reported.  
robm@homebox ~$ █
```



HEY — WHO DOES
SUDO REPORT THESE
"INCIDENTS" TO?

YOU KNOW, I'VE
NEVER CHECKED.



NICE NAUGHTY



FILESYSTEM: CHMOD

chmod

Sintassi: **chmod [-R] mode [pathname...]**

- **-R**: applica i permessi ricorsivamente
- **mode**: nuova maschera dei permessi
- **pathname**: oggetti a cui applicare la nuova maschera

Il comando **chmod** permette di cambiare i permessi dei file.

mode: sintassi numerica

`rw-r-----` → 110 100 000 → 640

`rwxr-xr-x` → 111 101 101 → 755

mode: sintassi simbolica

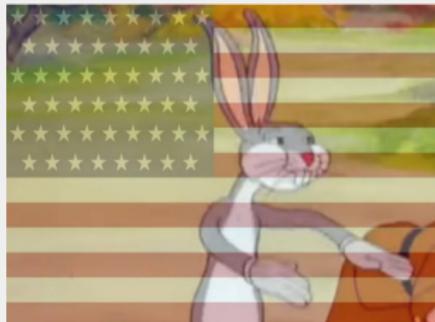
`rw-r-----` ↔ u=rw, g=r, o-rwx

`rw-rw-rw-` + u+x, o-rw → rwxrw----



chmod 777 file.txt

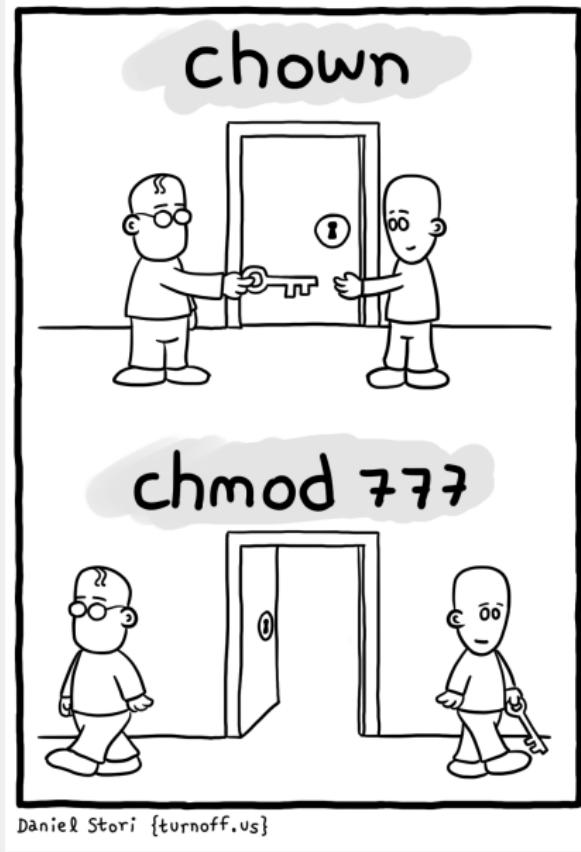
FILESYSTEM: CHOWN



chown

Sintassi: **chown [-R] owner[:group] [pathname...]**

- **-R**: esegue ricorsivamente le modifiche di proprietà
- **owner**: il nuovo proprietario
- **group**: il nuovo gruppo proprietario
- **pathname**: oggetti a cui vogliamo cambiare la proprietà



Daniel Stori {turnoff.us}

FILESYSTEM: LINK

I filesystem UNIX permettono di creare dei file “puntatori” ad oggetto del filesystem, questi sono detti **link**.

Soft link

I **soft link** (o **link simbolici**) sono dei file speciali che contengono al loro interno il pathname per raggiungere il **file target**. Chiaramente, se viene rimosso il file target, il soft link diventa inconsistente.

Hard link

Gli **hard link** (o **link fisici**) agiscono a livello del filesystem. In particolare, quando creiamo un hard link creiamo un puntatore all'**inode** presente nel filesystem.

Tale operazione è quindi trasparente a livello applicativo. Non è possibile, per ovvi motivi, creare hard link in filesystem differenti.

FILESYSTEM: LN / READLINK

ln

Sintassi: **ln [-s] target... [linkpathname]**

- **-s**: crea un link simbolico invece che uno fisico
- **target**: file o directory a cui il link farà riferimento
- **linkpathname**: pathname del link

Il comando **ln** consente di creare link tra i file e le directory presenti nel filesystem.

readlink

Sintassi: **readlink pathfile...**

- **pathfile**: il path del link simbolico da risolvere

Il comando **readlink** restituisce in output la risoluzione del link simbolico.

FILESYSTEM: FIND



find

Sintassi: **find [pathname...] [expression]**

- **pathname:** percorso in cui cercare ricorsivamente i file
- **expression:** le regole di selezione del file:
 - ▶ **opzione:** modifica il comportamento della ricerca
 - ▶ **condizione:** condizioni da verificare
 - ▶ **azione:** specifica cosa deve fare quando trova il file

Ricerca nei pathname i file che soddisfano le espressioni date

FILESYSTEM: FIND

Esempio

```
$ find . - name '*.sh' - print  
./script.sh  
./prova.sh  
./conta.sh  
./cartella/esercizio/eseguimi.sh
```

```
$ find /home/davide -name '*.bak' -exec rm
```

```
$ find /etc/ -type d -print /etc  
/etc/pacman.d  
/etc/pacman.d/gnupg  
/etc/pacman.d/gnupg/private-keys-v1.d  
/etc/pacman.d/gnupg/openpgp-revocs.d  
...
```

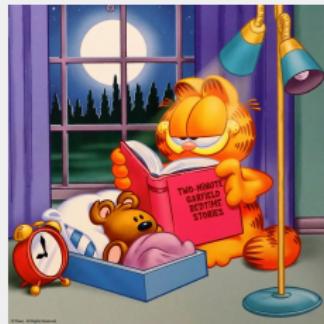


echo

Sintassi **echo [-n] [-e] [stringa...]**

- **-n**: non aggiunge l'endline al termine della stringa
- **-e**: permette l'uso di alcuni caratteri speciali
- **stringa**: stringa da visualizzare

Il comando **echo** da in output una stringa passata come parametro.



cat

Sintassi: **cat [pathname...]**

- **pathname:** file da visualizzare

Il comando **cat** permette di mandare allo standard output il contenuto di uno o più file.

MORE & LESS

more

Sintassi: **more [pathname...]**

- **pathname:** file da visualizzare

Il comando **more** si comporta come **cat** con l'eccezione che se l'output è più lungo di una videata di schermo, ad ogni pagina viene fatta una pausa.

less

Sintassi: **less [pathname...]**

- **pathname:** file da visualizzare

Il comando **less** è una versione più evoluta di **more**: non solo permette di fare pause ad ogni pagina, ma permette anche di andare su e giù nell'output. Inoltre è possibile effettuare ricerche interattive all'interno del file.

REDIREZIONE DELL'I/O

Ogni processo ha 3 flussi di dati:

- **standard input**: da cui prende il suo input; in genere corrisponde alla tastiera
- **standard output**: a cui invia il suo output; in genere corrisponde al terminale video
- **standard error**: a cui invia gli eventuali messaggi d'errore; anche qui si usa in genere il terminale

Attraverso l'invocazione da riga di comando è possibile redirezionare talu flussi su altri file.

- >: redireziona l'output su un file
- >>: redireziona l'output su un file in modalità **append**
- <: prende l'input da un file
- 2>: redireziona lo standard error su un file

REDIREZIONE DELL'I/O: ESEMPI

```
$ ls > output.txt
$ cat output.txt
esempio2.txt esempio3.txt esempio.txt

$ echo Ciao » output.txt
$ cat output.txt
esempio2.txt esempio3.txt esempio.txt
Ciao

$ cat < output.txt
esempio2.txt esempio3.txt esempio.txt
Ciao
$ man comandochenonesiste
Non c'è il manuale per comandochenonesiste

$ man comandochenonesiste 2> /dev/null
```

grep

Sintassi **grep [-i] [-l] [-v] [-w] [pattern] [filename]**

- **-i**: ignora le differenze tra maiuscole e minuscole
- **-l**: fornisce la lista dei file che contengono il **pattern**
- **-v**: stampa le linee che **non** contengono il **pattern**
- **-w**: vengono restituite le linee che contengono il **pattern** come parola completa
- **pattern**: espressione da ricercare
- **filename**: file da analizzare

Cerca all'interno delle righe dei file specificato le righe che contengono (o meno, a secondo delle opzioni) il pattern specificato. Il pattern può essere una semplice stringa o una regular expression.

ESPRESSIONI REGOLARI

Attraverso le **espressioni regolari** è possibile specificare dei *pattern* più complessi della semplice stringa contenuta.

metacarattere	significato
^	inizio della linea
\$	fine della linea
.	un singolo carattere qualsiasi
[str]	un qualunque carattere in str
[^str]	un qualunque carattere non in str
[a-z]	un qualunque carattere tra a e z
\	inibisce l'interpretazione del carattere
*	0 o più ripetizioni dell'elemento precedente
+	1 o più ripetizioni dell'elemento precedente
?	0 o una ripetizione dell'elemento precedente

GREP: ESEMPI

- **grep -F rossi /etc/passwd**

fornisce in output le linee del file che contengono la stringa **rossi**

- **grep -G -v '[agt]+'** relazione.txt

fornisce in output le linee del file che non contengono scritte composte dai caratteri **a, g, t**

- **grep -w print *.c**

fornisce in output le linee di tutti i file con estensione **.c** che contengono la parola **print**

- **grep -G '[a-c]+z'** doc.txt

fornisce in output le linee del file che contengono una stringa che ha un prefisso di lunghezza non nulla, costituito solo da lettere **a, b, c** seguito da una **z**



Due o più comandi possono essere messi in cascata collegando l'output del precedente con l'input del successivo. La sintassi è la seguente:

comando1 | comando2 | ... | comandon

I comandi vengono mandati in esecuzione contemporaneamente: ciascun processo aspetta che arrivi man mano l'output del precedente.

Esempio

```
ls /usr/bin | more
```

WC

Sintassi: **wc [-c] [-w] [-l] [pathname...]**

- **-c, -w, -l**: conteggia, rispettivamente, i caratteri/le parole (separate da spazi)/le righe
- **pathname**: file da analizzare

Il comando **wc** effettua l'analisi del suo standard input (se non vengono passati parametri) o dei file passati conteggiando il numero di byte, parole e/o righe. Se non si specifica una opzione particolare vengono riportati tutti e tre i conteggi.

SORT

sort

Sintassi: **sort [-n] [-r] [-o file] [-t s]
[-k s1[,s2]] [pathname]**

- **-n**: considera numerica la chiave di ordinamento
- **-r**: ordina in modo non crescente
- **-o file**: invia l'output su un file
- **-t s**: usa **s** come separatore
- **-k s1,s2**: usa i campi da posizione **s1** a **s2** per l'ordinamento
- **pathname**: file da ordinare

Il comando **sort** ordina trattando ogni linea del suo input come una collezione di campi separati da delimitatori. L'ordinamento di default avviene in base al primo campo ed è alfabetico.

HEAD & TAIL

head

Sintassi: **head [-c q] [-n q] [pathname...]**

- **-c q**: mostra solo i primi **q** byte dell'input
- **-n q**: mostra solo le prime **q** righe dell'input

Il comando **head** mostra di default le 10 righe del suo input.

tail

Sintassi: **tail [-c q] [-n q] [pathname]**

Il comando **tail** si comporta come **head** ma a partire dalla fine dell'input.

Esempio

```
cat /etc/passwd | tail -n 2 | head -n 1
```



Il comando **tar** ci permette di aggregare file e/o cartelle in un archivio (con estensione .tar).

Opzionalmente, è possibile comprimere gli archivi facendoli passare per un filtro di compressione (**bzip**, **bzip2**, **xz**) in modo esplicito (usando una pipe) oppure in modo implicito (utilizzando le opzioni **-z**, **-j** o **-J**).

TAR: SINTASSI

tar

Sintassi: **tar [-c|-x|-t] [-z|-j|-J] [-v] [-f archive] [pathname]**

- **-c**: crea un archivio
- **-x**: estraе un archivio
- **-t**: elenca il contenuto di un archivio
- **-v**: mostra il progresso (modalit **a verbose**)
- **-z/-j/-J**: comprime l'archivio con **gzip/bzip2/xz**
- **-f archive**: specifica l'archivio
- **pathname**: file e/o directory da comprimere

TAR: ESEMPI

```
$ tar -c -v -f prova.tar *.tex *.pdf images/
slides.tex
slides.pdf
images/
images/penguin.pdf
images/penguin2.pdf
...
$ tar x -f prova.tar
$ tar cjf prova2.tar.bz2 images/ *.pdf

$ bzcat prova2.tar.bz2 | tar tv
drwxr-xr-x davide/davide 0 2023-06-04 10:07 images/
-rw-r--r-- davide/davide 2336 2023-03-28 17:42 images/penguin.pdf
-rw-r--r-- davide/davide 1886 2023-03-28 17:44 images/penguin2.pdf

$ tar c images/ *.pdf | gzip > prova3.tar.gz
```

ALIAS

alias

Sintassi: **alias [name[=value]]**

Il comando **alias** permette di associare a nomi propri sequenze arbitrarie di comandi e opzioni. Inoltre,

- **alias** (senza parametri): restituisce la lista degli alias attualmente attivi
- **alias name**: restituisce l'associazione attuale per **name**
- **unalias name**: rimuove l'alias **name**

Esempio

```
$ alias ll='ls -l'  
$ ll /  
lrwxrwxrwx 1 root root 7 31 gen 21.51 bin -> usr/bin  
drwxr-xr-x 1 root root 180 31 mag 10.56 boot  
drwxr-xr-x 21 root root 4,1K 4 giu 08.04 dev  
...
```

MODALITÀ DI ESECUZIONE

Esistono varie modalità di esecuzione dei comandi sulla shell. La più semplice consiste nell'invocazione di un singolo comando. Ogni comando restituisce un **exit status** che rappresenta l'esito dell'esecuzione del comando stesso.

L'**exit status** è un intero nel range [0, 255].

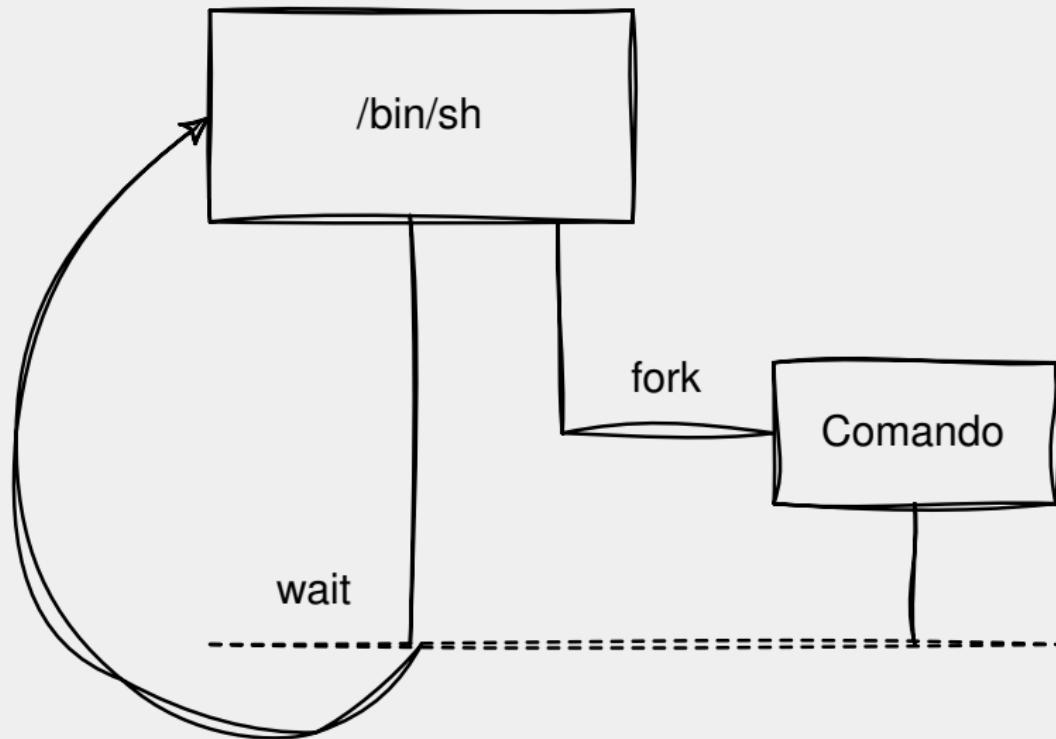
Questo è posto a:

0: se l'esecuzione è riuscita con successivo

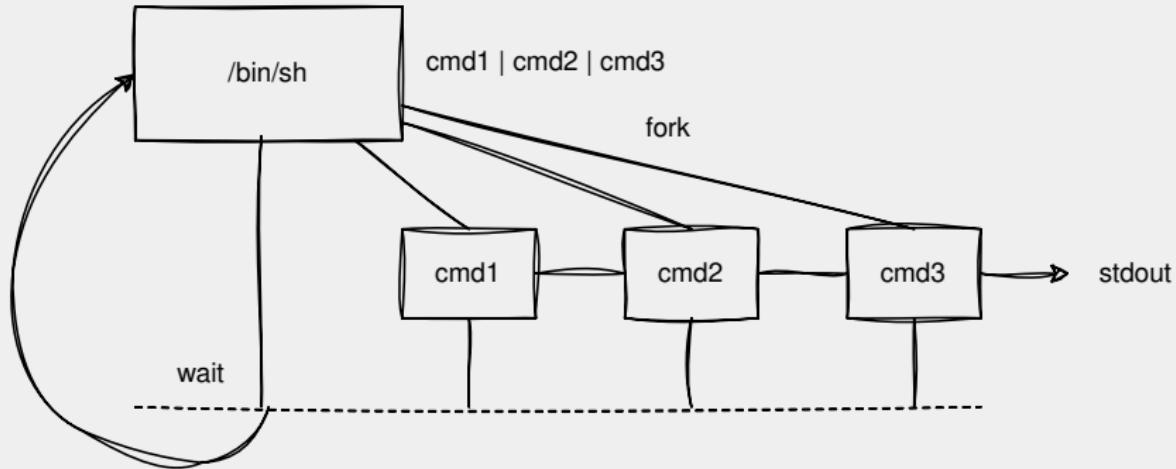
> 0: se l'esecuzione è fallita

I valori nel range [1, 255] sono utili per specificare all'utente il tipo di errore riportato durante l'esecuzione.

SIMPLE COMMAND EXECUTION



PIPELINE



LIST OF COMMAND

La lista di comandi permette di raggruppare una sequenza di comandi. La sintassi per l'invocazione è la seguente

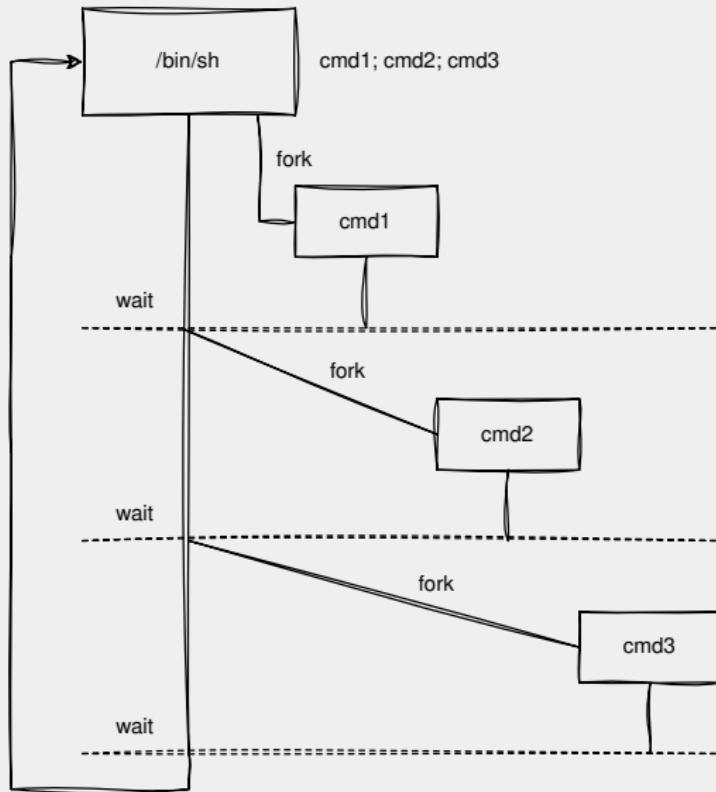
comando1 [; comando2...]

L'**exit status** della sequenza è uguale a quello dell'ultimo comando presente nella lista.

Esempio

```
$ echo -n "Via "; echo -n "del "; echo "campo"  
Via del campo
```

LIST OF COMMAND



LIST OF COMMAND

Nelle sequenze di comandi è possibile utilizzare gli operatori condizionali **&&** e **||**. La semantica è la seguente:

- **c1 && c2**: c2 viene eseguito se e solo se l'exit status di c1 è 0.
- **c1 || c2**: c2 viene eseguito se e solo se l'exit status di c1 è diverso da 0.

L'exit status di una lista condizionata è uguale all'exit status dell'ultimo comando eseguito.

Esempio

```
$ cat fileinesistente && echo fatto  
cat: fileinesistente: File o directory non esistente
```

```
$ cat fileinesistente || echo errore  
cat: fileinesistente: File o directory non esistente  
errore
```

ASYNCHRONOUS EXECUTION

Un comando può essere eseguito in **background** usando la sintassi

comando &

Inoltre, un processo in **foreground** può essere mandato in background interattivamente attraverso la combinazione di tasti **CTRL + Z**. In questo caso però il processo sospeso.

Questo può essere riportato in foreground utilizzando il comando **fg**.

CONTROLLO DEI JOB

La shell associa ad ogni comando un **job**. Ogni job viene identificato univocamente da un numero, detto **jobnumber**. Proprio per questo, quando un comando viene eseguito in modo asincrono, la shell restituisce il seguente output

[jobnumber] PID

Per controllare i job è possibile utilizzare i seguenti comandi:

- **jobs**: visualizza i job attivi e il loro stato
- **bg**: manda in background un job
- **fg**: porta in foreground un job
- **kill**: invia un segnale ad un job

BASH

BASH: INTRODUZIONE



BASH rappresenta l'evoluzione della Bourne shell (*sh*).

BASH è sia un interprete di comandi che un linguaggio di scripting, infatti è possibile utilizzare le utility che abbiamo visto fino ad ora con i costrutti tipici della programmazione.

BASH è la shell interattiva di default sulla maggior parte delle distribuzioni GNU/Linux e su Mac.

SCRIPT

Uno **script** è un insieme strutturato di comandi shell con strutture di controllo tipiche della programmazione imperativa.

Esempio di script

```
#!/bin/bash  
  
echo "Hello World"
```

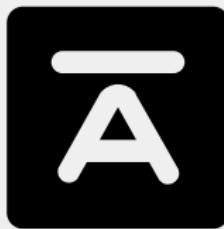
Esecuzione

Uno script può essere eseguito con la sintassi

bash nome-script.sh

Alternativamente, se questo possiede il permesso di esecuzione può essere eseguito direttamente con **./nome-script.sh**. In questo caso sarà il sistema operativo a passare il file all'interprete corretto, in base all'intestazione presente nel file.

VARIABILI



All'interno di uno script è possibile dichiarare delle variabili con la seguente sintassi

`var_name=value`

È possibile poi accedere al valore di una variabile utilizzando il metacarattere **\$** seguito dal nome della variabile:

`$var_name`

Di default le variabili dichiarate hanno *scope globale*.

VARIABILI BUILT-IN

- **\$#**: il numero di argomenti passati da command-line
- **\$i**: *i*-esimo argomento passato da command-line
- **\$***: lista di tutti gli argomenti passati da command-line (separati da \$IFS)
- **\$@**: lista di tutti gli argomenti racchiusi da doppi apici
- **\$?**: exit status dell'ultimo comando eseguito
- **\$!**: process id dell'ultimo comando eseguito in background

COMMAND SUBSTITUTION

È possibile assegnare l'output di un comando con il comando stesso con la seguente sintassi

`$ (comando)`

Risulta molto utile per inizializzare le variabili.

```
$ ls | wc -w  
15  
  
$ PAROLE=$(ls | wc -w)  
  
$ echo parole conteggiate: $PAROLE  
parole conteggiate: 15
```

QUOTING

Il meccanismo del **quoting** serve ad eliminare il metasignificato ad alcuni caratteri. Questo può essere di tre tipi:

- \ (*backslash*): elimina il metasignificato del carattere seguente
- '' (*single quote*): elimina il metasignificato da tutti i caratteri contenuti al suo interno
- "" (*double quote*): elimina il metasignificato da tutti i caratteri contenuti al suo interno ad eccezione di \$, ' e \.

Esempi

```
$ echo $PATH  
/usr/local/bin:/usr/bin:/usr/local/sbin...  
$ echo \$PATH  
$PATH  
  
$ echo "Il mio nome è $USER e ho 5$.  
Il mio nome è davide e ho 5$.
```

ESPRESSIONI ARITMETICHE



La valutazioni di espressioni aritmetiche con numeri interi si può ottenere tramite la seguente sintassi

$$((a=3+4/2))$$

La variabile *a* conterrà il risultato dell'espressione.

Nota

BASH non permette l'esecuzione di operazioni in virgola mobile, nel caso in cui fosse necessario è possibile invocare applicativi come *bc*, *calc* o *python*.

CONDIZIONI



In BASH è possibile esprimere condizioni attraverso le parentesi quadre:

[condizione]

Tali condizioni possono essere concatenate usando **&&** e **||**:

[cond1] && [cond2]

Nota

In realtà, la sintassi **[condizione]** rappresenta una forma compatta per l'esecuzione del comando *test* il cui *exit status* è posto a 0 se l'espressione è vera, 1 altrimenti.

CONDIZIONI: OPERATORI BINARI

Op. aritmetici	Op. stringhe	Significato
-eq	=	uguaglianza
-neq	!=	disuguaglianza
-lt	\<	minore
-le		minore o uguale
-gt	\>	maggiore
-ge		maggiore o uguale
	-z	stringa vuota
	-n	stringa non vuota

CONDIZIONI: OPERATORI SU FILE

Operatore	Significato
-e	Il file esiste
-f	Il file esiste ed è <i>regolare</i>
-d	Il file è una <i>directory</i>
-h/-L	Il file è un <i>soft link</i>
-O	Il file è di tua proprietà
-s	Il file è non vuoto

Operatore NOT

In generale è possibile negare una condizione anteponendo l'operatore **!**, ad esempio:

! -e file

restituisce *true* se il file non esiste, *false* altrimenti.

COSTRUTTO IF

```
#!/bin/bash

read -p "Inserisci un numero: " n

if [ $n -gt 10 ]
then
    echo "Il numero è più grande di 10"
elif [ $n -lt 10 ]
then
    echo "Il numero è più piccolo di 10"
else
    echo "Il numero è uguale a 10"
fi
```

COSTRUTTO IF (VERSIONE COMPATTA)

```
#!/bin/bash

read -p "Inserisci un numero: " n

if [ $n -gt 10 ]; then
    echo "Il numero è più grande di 10"
elif [ $n -lt 10 ]; then
    echo "Il numero è più piccolo di 10"
else
    echo "Il numero è uguale a 10"
fi
```

COSTRUTTO WHILE

```
#!/bin/bash

i=0

while [ $i -lt 10 ]
do
    echo -n "$i "
    ((i++)) # oppure i=$((i+1))
done

echo ""
```

COSTRUTTO FOR

```
#!/bin/bash

for w in frodo gandalf pipino; do
    echo $w
done

for i in {0..10}; do
    echo -n "$i "
done

echo ""

min=0
step=2
max=10

for i in $(seq $min $step $max); do
    echo -n "$i "
done

echo ""
```

COSTRUTTO: SWITCH

```
#!/bin/bash

read -p "Scegli tra (1) Frodo (2) Gandalf (3) Pipino: " scelta

case $scelta in
    "1"|"Frodo")
        echo "Hai scelto Frodo"
        ;;
    "2"|"Gandalf")
        echo "Hai scelto Gandalf"
        ;;
    "3"|"Pipino")
        echo "Hai scelto Pipino"
        ;;
    *)
        echo "Personaggio non riconosciuto."
        ;;
esac
```

ARRAY

```
#!/bin/bash

names=(Frodo Gandalf Pipino)

echo ${names[0]}
echo ${names[1]}
echo ${names[2]}

declare -a names2
names[0]=Frodo
names[1]=Gandalf
names[2]=Pipino

echo ${names[@]}
```

HELLO FUNCTION

```
#!/bin/bash

function hello() {
    [ $# -lt 1 ] && return 1
    [ -z "$1" ] && return 2
    local name=$1

    echo "Hello $name"
    return 0
}

hello function
echo "exit status: $?"

result=$(hello function)
echo $result
```

SUBSHELL

Una **subshell** è un processo figlio lanciato dalla *shell* (o da uno script).

Le subshell permettono di parallelizzare gli script, eseguendo sotto-task simultaneamente.

subshell-test.sh

```
#!/bin/bash
(
# Inside parentheses, and therefore a subshell . . .
while [ 1 ]; do
    echo "Subshell running . . ."
done
)

# Script will run forever,
#+ or at least until terminated by a Ctl-C.
exit $? # End of script (but will never get here).
```

RESTRICTED SHELL

Eseguire uno script o una porzione dello script in *restricted mode* permette di limitare i privilegi dello script al fine di minimizzare i possibili danni dovuti all'esecuzione dello stesso.

In particolare, uno script eseguito in questa modalità **non** può:

- Usare il comando *cd* per cambiare la working directory
- Modificare il valore delle variabili d'ambiente (*\$PATH*, *\$SHELL*, *\$ENV*, ...)
- Effettuare redirezione dell'output
- Invocare *exec*

RESTRICTED SHELL: ESEMPIO

```
#!/bin/bash

echo "Changing directory."
cd /usr/local
echo "Now in $(pwd)"
echo "Coming back home."
cd
echo "Now in $(pwd)"

# Everything up to here in normal, unrestricted mode.

set -r
# set --restricted has same effect.
echo "==> Now in restricted mode. <==""

echo "Attempting directory change in restricted mode."
cd ..
echo "Still in $(pwd)"
```



GET YOUR HANDS DIRTY!

ESERCIZIO 1

Scrivere uno script che prende in input n file di testo (*pathname*) e crea un file di testo di nome *merged.txt* che contiene l'unione di tutti i file di testo.

ESERCIZIO 2

Scrivere uno script che, presi in input il pathname di una directory contenente file di testo e una stringa s, restituisca in output il numero di righe contenenti la parola s all'interno dei file di testo.

Tip

Un file di testo può essere letto riga per riga utilizzando la seguente sintassi

```
while read line; do  
    echo "$line"  
done < file.txt
```