

Manuale dei comandi

Bash

Variabili e attribuiti

Creazione di una variabile

L'assegnazione per creare delle variabili formato da stringa:

```
a=1
```

Accedere al valore della variabile

Per accedere al valore della variabile si utilizza il carattere speciale "\$"

```
$a
```

NB: Da ricordarsi che viene interpretato da bash come comando e non come stampa della variabile

Cancellare una variabile

Tramite il comando "unset" si cancella la variabile:

```
unset a
```

NB: Ricordarsi di utilizzare il nome della variabile senza il \$, perchè non avrebbe senso, visto che non è necessario accedere alla variabile per cancellarla

Dichiarazione di una variabile intera

Tramite il comando "**declare**" e la sua opzione "-i"

```
declare -i a=1
```

ora verrà trattata come intera.

Stampa dei diversi attributi di una variabile

Tramite il comando "**declare**" e la sua opzione "-p"

```
# comando:
declare -p a

# output
declare -i a="1"

# \analisi di ogni componente dell'output\
# declare: prefisso output
# -i: attributo intero
# a: dichiarazione della variabile
```

Cancellare un attributo

Tramite il comando "**declare**" e la sua opzione "+i"

```
declare +i a
```

Lettura di un input

Tramite il comando "**read**" e il nome della variabile da leggere in input (cioè da tastiera):

```
read a
```

Scrittura in output

Tramite il comando "**echo**" si stampano le variabili su terminale

```
echo $a  
echo sono bello
```

Escaping

L'escaping è utile per stampare i caratteri speciali (che altrimenti vengono riconosciuti come tali)

```
echo \$a  
  
# output:  
$a
```

Quoting forte

Tramite il carattere ' (Apice singolo) è possibile incastrare un'espressione così da disabilitare qualunque meccanismo di interpretazione dei caratteri speciali

```
echo '$a$a$a$a$a$a$a$a$a$a'  
  
# output  
$a$a$a$a...
```

Altra caratteristica è il numero di argomenti utilizzati:

```
echo 'un argomento'  
echo due argomenti
```

L'intera espressione tra apici viene contato come singolo argomento.

Utility di sistema: printf

Tramite il comando "**printf**" si possono stampare dell'output formattato.

Il primo argomento è una stringa di formato C e ogni argomento successivo verra stampata come stringa:

```
printf '%s\n' stringa
print '%s\n' voglio essere bello
# output
stringa
voglio
essere
bello
```

Il primo argomento è formato da:

-%s: stampa in formato da stringa

-\n: vai a capo

Utility di sistema: touch

Tramite il comando "**touch**" crea file vuoti ed ogni argomento passato viene interpretato come nome di un nuovo file da creare:

```
touch file.txt

# creato file di nome file.txt
```

E' possibile abbinare dei nomi dinamici tramite il **quoting**
debole

Quoting debole

Tramite il carattere " (Apice dopppio) si effettua il quoting debole, chje consiste nell'incastrare una espressione, mantenendo l'interpretazione di alcuni caratteri speciali (i piu utilizzati in questo corso: \$ ' \)

```
f='nuovo'
touch "file $f.txt"

# output
# creazione di un nuovo file chiamata "file nuovo.txt"
```

Manipolazione

NB: Verra utilizzata, in questa sezione, la variabile `s=abcABC123ABCAbc` per capire molto bene il concetto di manipolazione.

Manipolazione nulla

Tramite l'espressione `${var}` si introduce il concetto di manipolazione, molto utile per evitare ambiguità nella rappresentazione delle variabili:

```
s=abcABC123ABCAbc
echo ${s}

# output
abcABC123ABCAbc
```

Per ambiguità si intende:

```
echo $sOK  
echo ${s}Ok  
## output
```

```
abcABC123ABCAbcOk
```

Uso di un valore di default

L'espressione `${var:-var}` effettua due operazioni importanti.

- Se `var` è già definita e non nulla, viene stampata con il suo valore
- Altrimenti viene stampata la stringa `"-var"`

```
echo ${s:-1}  
echo ${nonesistente:-1}  
echo $nonesistente
```

```
# output  
abcABC123ABCAbc  
1
```

NB: Questa espressione, come notato dall'esempio, non imposta nessun valore alla variabile

Impostazione di un valore alla variabile

L'espressione `${var:=var}` effettua due operazioni importanti:

- Se `var` è già definita e non nulla, stampa il suo valore
- Altrimenti, **var** viene definita e imposta il valore `"=val"`

```
echo ${s:=1}  
echo ${nonesistente:=1}  
echo $nonesistente
```

```
# output (in fila)
abcABC123ABCAbc
1
1
```

Lunghezza di una variabile

L'espressione `${#var}` stampa la lunghezza di una variabile:

```
echo ${#a}

# output
15
```

Estrazione di una sottostringa

L'espressione `${var:start}` estrae da `var` la sottostringa che inizia con l'indice **start**

```
echo ${s:0}
echo ${s:7}

# output
abcABC123ABCAbc
23ABCAbc
```

E' possibile anche scegliere la lunghezza della sottostringa con l'espressione `${var:start:len}`:

```
echo ${s:7:3}

# output
23A
```

Rimozione di una sottostringa (non greedy removal)

L'espressione `${var#pattern}` rimuove il match più piccolo all'interno della stringa.

Il pattern deve corrispondere all'inizio della stringa, senno non c'è alcun cambiamento nella stringa.

```
echo ${s#a*c}
```

```
# output
```

```
ABC123ABCabc
```

E' possibile rappresentarlo in due modi:

- Stringa esatta: "abc"
- Stringa con caratteri speciali di match: `(a*c)(a?c)`

Rimozione di una sottostringa (greedy removal)

L'espressione `${var##pattern}` rimuove il match più grande di pattern da inizio di var:

```
echo ${s###a*c}
```

```
# output
```

Qui tra i due candidati si sceglie il più lungo, che viene rimosso.

Rimozione di una sottostringa partendo dalla fine

Le espressioni `${var%pattern}` e `${var%%pattern}` rimuovono rispettivamente match più piccolo (non greedy) e il match più grande partendo dal fondo

NB: Pattern e stringa confrontati sempre da sinistra verso destra

```
echo ${s%a*c}
```

```
# output
```

```
abcABC123ABC
```

Operazioni matematiche

Operatore per espressioni aritmetiche

L'operatore `$((EXPR))` interpreta l'espressione aritmetica `EXPR` e ne restituisce il risultato

E' possibile utilizzare degli **operandi costanti**, **valori di variabili** e infine è possibile sia **salvare il risultato in una variabile**

```
a=1
```

```
b=2
```

```
echo $((1+2))
```

```
echo $((a+b))
```

```
var=$((1+2))
```

```
echo $var
```

```
# output
```

```
3
```

```
3
```

```
3
```

Statement let

Lo statement **let** è uguale all'assegnazione classica, ma più elegante:

```
a=1
b=2
let c=a+b
echo $c

# output
3
```

Ambiente

Le variabili di ambiente configurano il comportamento delle applicazioni, possono essere **builtin** (interno al bash) oppure **esterne** (impostate esternamente dall'utente per specificare applicazioni esterne).

Tramite il comando **env** si stampano tutte le variabili contemporaneamente

Per vedere tutte le variabili [Parte 5 - BASH > Ambiente](#)

Operatori condizionali e logici

Ecco una lista della sintassi per ciascun operatore condizionale e logico:

```
# CONFRONTO ARITMETICO
ARG1 -eq ARG2 # VERO se ARG1=ARG2
ARG1 -ne ARG2 # VERO se ARG1!=ARG2
ARG1 -lt ARG2 # VERO se ARG1<ARG2
ARG1 -gt ARG2 # VERO se ARG1>ARG2
ARG1 -le ARG2 # VERO se ARG1<=ARG2
ARG1 -ge ARG2 # VERO se ARG1>=ARG2
```

```

# CONFRONTO TRA STRINGHE
-z STR # VERO se STR ha lunghezza zero
-n STR # VERO se STR è non nulla
STR # VERO se STR non ha lunghezza zero
STR1 == STR2 # VERO se STR1 e STR2 sono uguali
STR1 != STR2 # VERO se STR1 e STR2 non sono uguali
STR1 \< STR2 # VERO se STR1<STR2
STR1 \> STR2 # VERO se STR1>STR2

# CONFRONTO BOOLEANO
EXPR # ritorna il valore logico di EXPR
! EXPR # VERO se EXPR ha valore FALSO
EXPR1 -a EXPR2 # VERO se EXPR1 e EXPR2 lo sono
EXPR1 -o EXPR2 # VERO se EXPR1 o EXPR2 lo sono

```

nb: 0 - VERO, qualunque numero sopra lo 0 - FALSO

Test di una condizione e risultato

Tramite il comando **test** è possibile testare una operazione logica tra parentesi quadre (o senza).

Il risultato viene memorizzato nella variabile speciale \$?, che salva il risultato del test.

```

test [ 1 -lt 2]
echo $?

# output
0 # vero

```

Controllo di flusso (costrutti if, case, while, until, for, do-while).

Bash è **imperativo**, il controllo del flusso è esplicitato dall'utente tramite costrutti

Costrutto if

La sintassi del costrutto if:

```
# comandi
if COND_TEST1; then
    STATEMENTS1
elif COND_TEST2; then
    STATEMENTS2
else
    STATEMENTS3
fi
# spiegazione:
# Il costrutto è delimitato dalle stringhe if e fi
# Se la prima condizione è corretta, fa lo statements1 e
# poi esce, invece se nel caso non dovesse essere corretta,
# usa il then e va nella seconda condizione e così via.

# i token "then" separano i test condizionali dagli
# statements associati
```

Per l'inserimento multilinea va usato il backslash per spezzare il comando nella riga successiva ed ad ogni statement va messo il ";"

```
if [ $a -ge 0 -a $a -le 10 ]; then \
    echo "a è compreso tra 0 e 10"; \
elif [ $a -ge 11 -a $a -le 20 ]; then \
    echo "a è compreso tra 11 e 20"; \
else echo "a non è compreso tra 0 e 20"; \
fi
```

Costrutto case

La sintassi generale di "case":

```
# comandi
case $var in
VAL1)
    STATEMENTS1
;;
VAL2)
    STATEMENTS2
;;
VAL3)
    STATEMENTS3
;;
esac

# spiegazione:
# IN e ESAC: sono le stringhe che delimitano il costrutto.
# $var in: $var è la variabile che serve per la ricerca ,
# il token in separa la variabile dall'elenco dei valori
# var==VAR1: Entra nel primo statement poi esce e così via
```

Al posto di **VAR** è possibile mettere dei caratteri speciali come "*" che matcha ogni singola possibilità (come il default nel linguaggio di programmazione)

Costrutto while

Sintassi generale

```
while COND_TEST1;
do
    STATEMENTS1
done
```

```
# spiegazione
# do e done: fanno da corpo per il ciclo
# COND_TEST1: è un test condizionale e il token ";" separa
la condizione dal corpo del ciclo
# Ovviamente entra nel while se la condizione è true,
esegue il corpo. sennò esce.
```

Costrutto until

Sintassi generale

```
until COND_TEST1;
do
    STATEMENTS1
done
# spiegazione
# do e done: fanno da corpo per il ciclo
# COND_TEST1: è un test condizionale e il token ";" separa
la condizione dal corpo del ciclo
# Qui funziona all'inverso del while, qui finch è falso
esegue lo STATEMENTS1 sennò esce.
```

Costrutto for

Sintassi generale

```
for var in LIST;
do
    STATEMENTS1
done
```

Nella parte "LIST" si inserisce un elenco di stringhe, scritte in fila.

Invece "var" è una variabile che riceve il valore i-mo di LIST fino all'ultimo valore per il ciclo.

In "LIST" è possibile utilizzare un'espansione tramite le graffe per rappresentare un insieme di numeri:

- {0..N} stampa i numeri da 0 a N
- {0..N..M} stampa i numeri da 0 a N a passi di M (per es M=2, 0 2 4 6 8 .. N)
- $S_1 \{S_2, S_3\} S_4$ viene composto un gruppo con S_1, S_2, S_4 e uno con S_1, S_3, S_4

Manipolazione delle iterazioni

Esistono i comandi "break" e "continue" che funzionano uguale al linguaggio di programmazione Java, C#, madonna cane:

- **Break**: esce immediatamente dal costrutto di controllo del flusso corrente
- **Continue**: esegue immediatamente la prossima iterazione del costrutto di controllo del flusso corrente

Esecuzione condizionata

L'operatore ";" **consente di concatenare più comandi** che vengono eseguiti **rigorosamente uno dietro l'altro**, indipendentemente da terminazioni corrette o non

```
# comandi
ls nonesistente; echo prova

# output
ls: impossibile accedere a 'nonesistente': ...
prova
```

L'operatore "&&" esegue il **comando successivo solo se il precedente non risulta nulla**

```
# comandi
# 1
ls nonesistente && echo "tutto ok"

# 2
ls /bin/bash && echo "tutto ok"
# output
# 1

# 2
tutto ok
```

L'operatore "||" (doppia pipe) esegue il comando successivo solo se il precedente esce con **stato nullo**

```
# comandi
# 1
ls nonesistente || echo "errore"
# 2
ls /bin/bash || echo "Errore"
# output
# 1
Errore
# 2
/bin/bash
```

Script

Creazione script

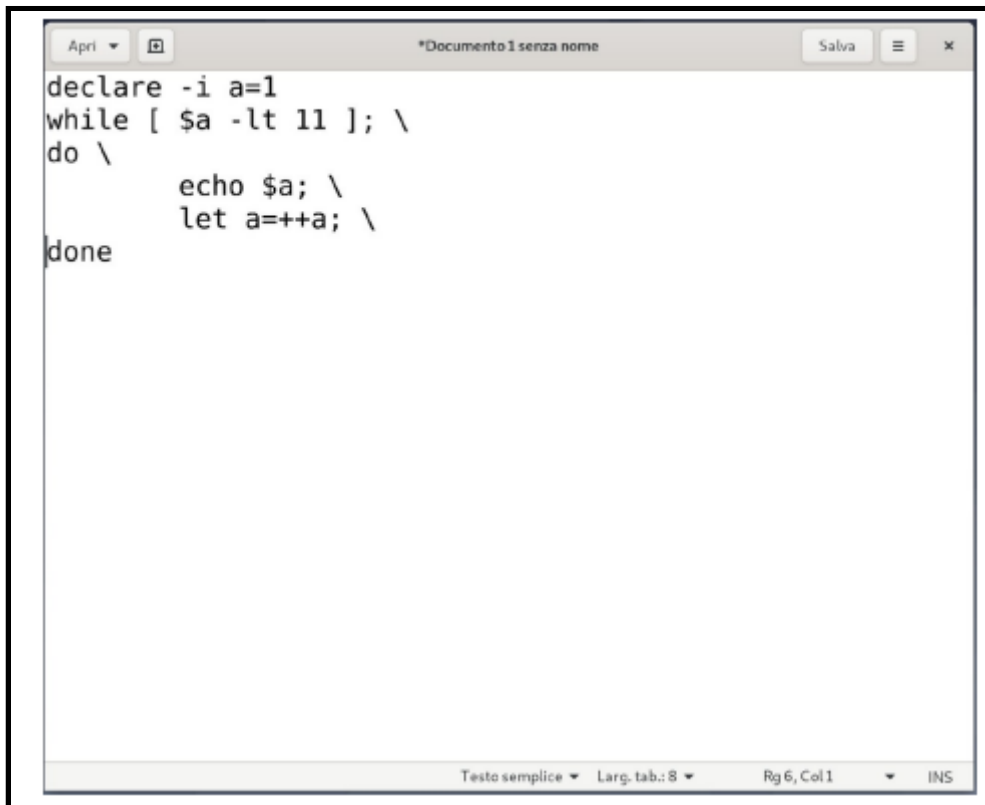
E' possibile creare degli script tramite un editor di testo, per esempio GEDIT su GNOME.

Scriviamo il codice che ci serve all'interno del file e infine lo salviamo "filename.sh".

Per avviarlo basta eseguire "bash filename.sh" e da lì funziona lo script.

Formattazione dello script

- Tutti comandi su una riga (per uccidersi)
- Ogni statement su una riga, senza multiline (per uccidersi)
- Ogni statement su una riga con multiline (il migliore): gli statement del corpo del ciclo devono terminare con ";".

A screenshot of a text editor window titled "*Documento 1 senza nome". The window contains a shell script with the following code:

```
declare -i a=1
while [ $a -lt 11 ]; \
do \
    echo $a; \
    let a=++a; \
done
```

The code is formatted with backslashes at the end of each line to allow for multiline statements. The editor interface includes a menu bar with "Apri", "Salva", and a close button. The status bar at the bottom shows "Testo semplice", "Larg. tab.: 8", "Rg 6, Col 1", and "INS".

multilinee ogni statement su una riga.

Parametri posizionali

Esistono delle variabili interne già esistenti (builtin) che vengono chiamate parametri posizionali.

Consistono in delle stringhe che un utente fornisce ad uno script, al fine di definirne la modalità operativa e l'oggetto delle operazioni

\$0: nome dello script
\$1: primo argomento
\$2: secondo argomento
e così via

```
# All'interno di uno script chiamato "stampa.sh"
echo "Nome script: $0"
echo "Parametro $1"

# all'interno del terminale
bash stampa.sh valore

# output
Nome script: stampa.sh
Primo argomento: valore
```

Segnalazione stato d'uscita

Tramite il comando **exit** è possibile segnalare lo stato d'uscita di uno script che poi è possibile visualizzare tramite il comando speciale `$?`

```
# all'interno di uno script chiamato "uscita.sh"
echo "operazione fallita"
exit 1

# all'interno di un terminale
bash uscita.sh
echo $?
# output
Operazione fallita
1
```

Debugging

Tramite il comando "bash" e qualche sua opzione è possibile effettuare del debugging stampando ogni statement prima della loro esecuzione.

-L'opzione "-v" stampa gli statements:

```
# comandi
bash -v conta.sh #script che conta i numeri da 1 a 3 e li
stampa

# output
declare -i a=1
while [ $a -lt 3 ]; do
    echo $a
    let a=++a
done
1
2
```

-L'opzione "-x" abilita l'esplicitazione di ogni statement prima della sua esecuzione così da capire se variabili e parametri sono espansi correttamente.

```
# comandi
bash -x conta.sh

# output
+ declare -i a=1
+ '[' 1 -lt 3 ']'
+ echo 1
1
+ let a=++a
+ '[' 2 -lt 3 ']'
+ echo 2
2
```

```
+ let a=++a  
+ '[' 3 -lt 3 ']'
```

```
# tutte le righe precedute da + sono dovute all'opzione -  
x, le altre righe sono l'output delle righe sopra
```

Alias e funzioni

Creazione di un alias

Per alias si intende una **abbreviazione di un comando**, lo si usa per statement particolarmente complessi.

Lanciando il comando "alias" si ottiene l'elenco degli alias disponibili

Per la creazione di un alias:

```
alias NOME_ALIAS = 'COMANDO'  
# esempio  
alias l='ls -hog'
```

Cancellazione di un alias

Tramite il comando "**unalias**" si effettua una cancellazione di un alias:

```
unalias NOME_ALIAS  
# esempio  
unalias l
```

Funzioni

Funzioni come nei linguaggi di programmazioni classici tranne che nelle parentesi tonde della funzione non possono essere

messe dei parametri.

Per la sintassi generale:

```
function FUNC_NAME()  
{  
    STATEMENTS  
}  
  
# spiegazione  
# function: è la parola chiave che introduce il costrutto  
(può essere omessa).  
# FUNC_NAME: nome della funzione  
# La coppia di parentesi è ornamentale. I parametri  
formali NON vengono scritti lì dentro  
# le graffe delimitano il blocco  
# un corpo di una funzione non può essere vuoto
```

Invocazione di una funzione

Per invocare una funzione basta scrivere il nome della funzione e gli eventuali argomenti:

```
FUNC_NAME arg1 arg2 arg3 ... argN
```

Per prendere gli argomenti basta utilizzare i parametri posizionali (\$1, \$2 ... \$n).

NB: \$0 ritornerà sempre il nome dello script e non il nome della funzione

Ritorno di un valore

Tramite il comando "return" permette di impostare la variabile speciale \$? con un valore che rappresenta lo stato di uscita.

```
return 1  
return "a"
```

Campi di visibilità e variabili locali/globali

Bash ha come campo di visibilità di default **globale** (E' una variabile in cui il **campo di visibilità** è l'intero programma), anche per le variabili all'interno delle funzioni.

E' possibile forzare una variabile per renderla **locale** tramite il comando "local":

```
local a=1
```

NB: La definizione locale è **possibile** solo all'interno delle **funzioni**

Invece come campo di visibilità delle variabili è di scope dinamico (Il campo di visibilità di una variabile locale è calcolato a tempo di esecuzione (run time, si coinvolgono blocchi di codice invocati a cascata))

Storia dei comandi e manipolazione

Cronologia comandi

La storia dei comandi è salvata **temporaneamente** all'interno di un buffer. Poi a **fine sessione di lavoro**, bash salva tutto all'intero di **un file di log**

- Tramite il comando "**history**", lanciato **senza argomenti** elenca tutti i comandi presenti nella cronologia (uno per riga)

- Lanciato con un **parametro N** (intero positivo) elenca gli ultimi N comandi nel **buffer**
- **Lanciato con l'opzione "-c", cancella il buffer (ma non il file di log** che tiene salvato tutti i comandi lanciati)

Si può anche forzare la sincronizzazione tra buffer e file di log prima della termine della sessione:

- Tramite l'opzione "-w", il buffer viene salvato sul file
- Tramite l'opzione "-a", il buffer viene appeso al file

Manipolazione dei comandi

Tramite il comando "fc" è possibile modificare la cronologia dei comandi.

- Avviato senza opzioni, aprirà un editor di default (GNU nano), il buffer dell'editor contiene l'ultimo comando ed è associato ad un file temporaneo
- Lanciato con un argomento (-N intero negativo) è possibile editare il comando N-mo (LAST-N = comando scelto, nella quale LAST è l'ultimo comando nella lista.)
- Lanciato con un argomento (N positivo) è possibile editare un comando a scelta.

- Lanciato con un argomento (S, sottostringa di un comando) matcha l'ultimo comando che contiene la sottostringa S
- Lanciato con un gruppo di argomenti(N1 e N2, due indici riferiti ai comandi) prende tutti i comandi presenti tra i due indici.

Tramite l'opzione "-l" del comando "fc" è possibile elencare i comandi della cronologia

- Lanciato senza argomenti, vengono stampati i primi 16 comandi
- Accetta tutti i tipi di argomenti visti sopra

Tipologia di comandi

Comando type

Tramite il comando "type", che riceve in ingresso uno o più parametri che rappresentano i comandi, ritorna la tipologia ad essa associata (che può essere: COMANDO INTERNO caricata da una funzione associata o COMANDO ESTERNO - fornito in una directory path)

```
# comandi
type help ls vmstat

# output
help è un comando interno di shell
```



```
ls ha "ls --color=auto" come alias  
vmstat è /usr/bin/vmstat
```

- Tramite l'opzione "-a" è possibile segnalare tutte le omonimie di un comando:

```
type -a printf  
  
# output  
printf è un comando interno di shell  
printf è /usr/bin/printf  
printf è /bin/printf  
# spiegazioni  
# sono presenti due percorsi /bin e /usr/bin per motivi di  
retrocompatibilità (sparirà poi negli aggiornamenti  
futuri), visto che ormai le distribuzioni GNU/Linux  
unificano i due binari. (di conseguenza i due comandi sono  
identici)
```

Documentazione

Builtin

Il comando help

Tramite il comando "help" stampa la sinossi (in formato Backus-Naur) di tutti i comandi interni forniti da bash

```
help
```

Passandogli come argomento un comando, "help" fornisce la documentazione di uno specifico comando all'interno di bash

```
help help
```

Come argomento è possibile utilizzare un pattern (*) sta ad indicare che darà tutti gli help dei comandi interni che iniziano con un determinato pattern

```
help "a*"
```

Ed è possibile passargli piu pattern mischiandoli.

```
help "a*" "b*" cd
```

Comandi esterni

Il comando man

Spiegazione

Tramite il comando "**man**" si ha la possibilità di vedere la documentazione dei comandi esterni al bash.

E' un vero e proprio libro di nove capitoli (o sezioni), riguardante diversi argomenti, per visionarli:

```
man man
```

Il comando man viene eseguito con un argomento che rappresenta il comando da visualizzare:

```
man ls
```

NB: Essendo un manuale abbastanza voluminoso, interviene un **paginatore**

E' possibile che una voce abbia piu sezioni e di conseguenza è possibile inserirla come argomento:

```
man 3 printf
```

```
# partirà dalla 3 sezione
```

(Ovviamente senza nessun specifica, partirà dalla prima)

Opzioni

- Tramite l'opzione "-S" o "-s" seguite da un intero (oppure tramite un elenco di sezioni separate dai due punti o dalla virgola) per selezionare una sezione/i

```
man -s/-S 3 printf
```

```
man -s 1:3 printf
```

- Tramite l'opzione "-a" si stampa tutte le voci presenti.
- Tramite l'opzione "--regex" permette di usare una espressione regolare per la sezione di pagine arbitrarie

```
man -a -S 2 --regex '.*'
```

NB: Il comando nell'esempio stampa tutte le pagine di manuale della sezione 2 perchè: il punto indica un qualunque carattere invece l'asterisco indica la ripetizioni (0 ... N volte) del carattere precedente.

Il comando apropos

Tramite il comando "apropos" è possibile ricercare delle informazione sull'esistenza di comandi con nomi esatti a

partire da chiavi di ricerca testuali

Il comando accetta un elenco di chiavi di ricerca, effettua una ricerca di tipo "OR" sui nomi delle voci

```
apropos KEY1, KEY2, ... KEYN
```

Opzioni

- Tramite l'opzione "-s" è possibile limitare la sezione della ricerca
- Tramite l'opzione "-a" effettua una ricerca di tipo "AND".

File

Gestione metadati

Lettura metadati

Tramite il comando "stat" è possibile visualizzare i metadati di un file:

```
stat [opzioni] nome_file
```

NB: si può utilizzare su tutti i file

Stampa un insieme di informazioni molto importanti, per visionarli: [Parte 7 - File > <u>Lettura dei metadati</u>](#)

Visione contenuti di una directory

Tramite il **comando** "ls" si visualizza il contenuto di uno/piu file o directory FD_1, FD_2, FD_N :

```
ls [opzioni] FD1, FD2, ... ,FDN
```

Opzioni

- Tramite l'opzione "-a" è possibile **visualizzare dei file/directory nascoste**, saranno contraddistinte dal nome del file che inizierà con un puntino (.nomefile)

```
ls -a
```

- Tramite l'opzione "-l" è possibile attivare la visualizzazione "lunga", cioè la stampa dei metadati principali per ciascun file (tipo di file/permessi, numero di collegamenti fisici, utente creatore, gruppo di lavoro, peso, tempo di accesso, nome)

```
ls -l
```

- Tramite l'opzione "-h" è possibile visualizzare le dimensioni dei file in formato "umano" cioè utilizzando le unità di misura informatiche (M, K, G) (si usa di solito con l'opzione "-l")

```
ls -l -h
```

- Tramite l'opzione "-t" si effettua una stampa in ordine di tempo d'accesso decrescente:

```
ls -l -t
```

- Tramite l'opzione "-S" ordina i file per dimensione decrescente.

```
ls -l -S
```

- Tramite l'opzione "-r" inverte l'ordinamento imposto con le opzione "-t" e "-S":

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

- Tramite l'opzione "-R" elenca ricorsivamente i file e le sottodirectory

```
ls -R
```

Identificazione del tipo di file

Tramite il comando "file", possiamo stampare la tipologia di uno/piu file o directory FD_1, FD_2, \dots, FD_N :

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

Per vedere il tipo di file di /etc/passwd, si esegue il comando:

```
# comando
file /etc/passwd

# output
/etc/passwd: ASCII text
```

Creazione e rimozione

Creazione di un file

Tramite il comando "**touch**", lanciato senza opzioni e con un elenco di nomi di file non esistenti, crea file con contenuto nullo e timestamp riferito all'istante attuale:

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

Opzioni

- Tramite l'opzione "-a" su un file esistente, ne altera il timestamp di ultimo accesso

```
touch -a filename
```

- Tramite l'opzione "-m" su un file esistente, ne altera il timestamp di ultima modifica:

```
touch -m filename
```

Creazione di directory

Tramite il comando "**mkdir**", si possono creare delle directory vuote:

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

Opzioni

- Tramite l'opzione "-p" risolve i diversi problemi che riguardano il comando mkdir, come la creazione di directory già esistente e la creazione di sottodirectory

Cancellazione di directory

Tramite il comando "rm", non si può cancellare nulla, ma necessita delle sue opzioni

Opzioni

- Tramite l'opzione "-d" è possibile cancellare directory vuote

```
rm -d dir
```

- Tramite l'opzione "-r" abilita la modalità di cancellazione ricorsiva, cioè la cancellazione della directory e del suo contenuto.

```
rm -i dir
```

- Tramite l'opzione "-i" abilita la modalità interattiva (cioè per ogni cancellazione viene chiesta la conferma all'utente)

```
rm -i dir/file
```

- Tramite l'opzione "-f" abilita la modalità forzata per la cancellazione di directory senza la conferma all'utente

```
rm -f dir
```

Alternativa a rm

Esiste un'alternativa al comando sopracitato che si chiama "**rmdir**" che tramite l'opzione "-p" consente di cancellare una gerarchia di sottodirectory (vuote):


```
rm -p dir1/dir2/dir3
```

Copia e spostamento

Copia di un file

Tramite il comando "**cp**" copia file e/o directory, son necessari due argomenti che sono F_{src} e F_{dst} che già dai nomi capiamo la loro utilità:

```
cp F_SRC F_DST
```

Copia di piu file in una directory

Sempre tramite il comando "cp" è possibile copiare piu file in una directory.

Se i primi N argomenti sono file F_1, F_2, \dots, F_N e l'ultimo argomento è una directory D , cp copia tutti i file all'interno della directory:

```
cp F1 F2 ... FN D
```

Opzioni

- Tramite l'opzione "**-a**" abilita la modalità di copia archiaviale, cioè la preservazione dei metadati originali di tutti i file passati in un'altra directory

Spostamento file/directory

Tramite il comando "**mv**" sposta file e/o directory. E' necessario due argomenti che sono FD_{src} e FD_{dst} , così da spostare il file da un punto ad un altro:

```
mv FDsrc FDdst
```

E' possibile anche cambiare nome durante lo spostamento, basta semplicemente usare un nome diverso nel FD_{dst} :

```
mv filename /tmp/filename_doppio
```

Opzioni

- Tramite l'opzione "-b" è possibile creare un backup durante lo spostamento di un file. (Il file di backup verrà contraddistinto da una tilde alla fine "filename2.txt~").

```
mv -b filename1.txt filename2.txt
```

Visualizzazione di un file

Tramite il comando "cat" è possibile visualizzare in maniera non interattiva un file. Viene lanciato senza opzioni e con un elenco di nomi di file esistenti F_1, F_2, \dots, F_N :

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

Opzioni

- Tramite il comando "-E" è possibile stampare i caratteri di fine linea con il simbolo "\$"

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

```
# output
```

```
root:x:0:0::/root:/bin/bash$
```

```
...
```

- Tramite l'opzione "-T" stampa il carattere tabulazione con il simbolo "^|"

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

- Tramite l'opzione "-v" è possibile stampare i **caratteri non stampabili** (per vedere i caratteri non stampabili basta guardare [Parte 7 - File > I caratteri non stampabili](#) e [Parte 7 - File > Storia della nascita di "META" e del codifica multi-byte](#))
- Tramite l'opzione "-n" stampa i numeri di riga

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

Visualizzazione di file binari

Tramite il comando "**hexdump**" stampa il contenuto di file in diversi formati.

Lanciato senza opzioni e con un elenco di nomi di file esistenti

$F_1 F_2 \dots F_N$ stampa il contenuto del file in esadecimale, a parole di 16bit:

```
hexdump F1 F2 ... FN
```

Un esempio, visualizziamo il contenuto in hex del file /etc/hostname:

```
# comandi
hexdump /etc/hostname

# output
000000 | 6564 6962 6e61 000a
000007 |

# spiegazione
# Nella parte di sinistra della pipe è presente l'offset
in esadecimale, a destra i byte in esadecimale
("debian\n")
# La pipe inserita serve solo per dividere i due
risultati, non esce nel risultato dell'output
```

Notiamo come i byte componenti le parole rappresentano a due a due le parole "Debian" rovesciate, cioè sarebbe "ed ib na", **come mai questo effetto?**

L'output di default di **hexdump** è a parole **esadecimali di 16** bit. Poichè l'architettura Intel è **Little Endian**, i numeri sono rappresentanti **con i bit meno significativi per primi** quindi da ASCII 0x64 e ASCII 0x65 **vengono invertiti nella versione Little Endian e stampati.**

Opzioni

- Tramite l'opzione "-C" abilita la modalità canonica cioè la combinazione dell'output in hex e nel formato stringa

```
# comandi
hexdump -C /etc/hostname

# output
Offset      | Dump esadeci. | dump_stringa |
00000000    | <16 byte hex> | <16 caratteri>|
00000010    | ...
```

- Tramite l'opzione "-v" attiva la modalità verbosa cioè non effettua nessuna sostituzione e inserisce anche le ripetizioni:

```
# comandi
hexdup -v /bin/ls

# output
0000240 0000 0000 0000 0000 0000 0000 0000 0000
0000250 0000 0000 0000 0000 0000 0000 0000 0000
0000260 0000 0000 0000 0000 0010 0000 0000 0000
```

Visualizzazione parte iniziale

Tramite il comando "**head**" visualizza la parte iniziale di un file (cioè le prime 10 righe):

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

Opzioni

- Tramite l'opzione "-n", riceve un argomento N, che corrisponde al numero di righe che si vuole stampare:

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

Esempio: stampa solo della prima riga

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

Esempio: stampa solo dell'ultima riga

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

- Tramite l'opzione "**-c**", concettualmente simile a **-n**, è possibile stampare i byte:
 - **C positivo**: stampa i **primi C byte**
 - **C negativo**: stampa fino agli **ultimi C byte**

Esempio: stampa dei primi dieci byte

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

Esempio: stampa dal primo al terzultimo

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

Visualizzazione parte finale

Tramite il comando "tail" è possibile visualizzare la parte finale di un file (duale a head, uguale anche le opzioni)

Opzioni

- Tramite il comando "**-n**", riceve in argomento **N**, nella quale:

-n N: stampa le ultime N righe
-n +N: stampa a partire dalla riga N

- Tramite il comando "-c", contettualmente uguale a -n, stampa i byte:
 - c C: stampa gli ultimi C byte
 - c +C: stampa a partire dal byte C
- Tramite l'opzione "-F" è possibile visualizzare nel terminale una specie di visualizzazione di file di log che svolge le seguenti operazioni:
 - Continua a provare l'apertura del file fino a quando non viene creato
 - mostra le ultime righe del file
 - si blocca in attesa di nuove righe da mostrare

Ricerca destinazione

Tramite il comando "**which**" è possibile riuscire a trovare la directory di un file:

```
which filename
```

Manipolazione testi

Selezione di elementi nel testo

Tramite il comando "cut" è possibile estrarre elementi selezionati da uno piu file esistenti:

```
cut [opzioni] F1 F2 ... FN
```

Opzioni

- Tramite l'opzione "-f" permette di **selezionare i campi all'interno di un file** e l'argomento viene utilizzato per creare una specifica selezione. (NB: Il carattere di delimitazione di default è <TAB>, per cambiarlo "-d")

```
#comandi  
cut -f 1 file.txt
```

Per selezionare i primi due campi di un file con separazione una tabulazione:

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

Per selezionare solo il primo e terzo campo di un file con separazione una tabulazione:

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

- Tramite l'opzione "-c" è possibile specificare un intervallo di caratteri e l'argomento viene usato per specificare un'intervallo di caratteri:
- Tramite l'opzione "-b" è possibile specificare un intervallo di byte e l'argomento viene usato per specificare un'intervallo di byte

Creazione di file multicolonnari (domanda esame fatta)

Tramite il comando "**paste**" fonde uno o più file esistenti $F_1 F_2 \dots F_N$ in un unico output multicolonnare:

```
paste f1 f2 ... fn
```

Un **esempio di utilizzo per vederne il funzionamento**:

1- Creiamo un file "id.txt" contenente una sequenza di numeri da 1 a 10 (uno per riga)

2- Creiamo un file "users.txt" contenente i primi dieci username in /etc/passwd

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

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

Unendo così le due colonne

Ordinamento di file per colonne

Tramite il comando **sort** ordina file secondo specifici criteri e i criteri vengono dati dalle opzioni

Opzioni/criteri

Esistono diversi criteri di ordinamento gestiti da sort e i più comuni sono;

-n: ordinamento numerico

-d: ordinamento alfanumerico (dizionario)

-h: ordinamento numerico "umano" (confronta 2G o 1K, unità di misura umane)

-M: ordinamento del mese (confronta JAN, FEB, ...)

Ordinamento in base a chiave specifica

E' possibile ordinare le righe di un file in base ad una chiave di ordinamento in essa contenuta

Tramite l'opzione "-k" che riceve come argomento una specifica testuale **KEY** del formato di una chiave, che illustra come recuperarla da una riga.

Tramite l'opzione "-t" specifica il carattere separatore dei campi

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

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

- **F** (obbligatorio): è il numero di campo desiderato
- **C** (facoltativo): è la posizione del carattere nel campo
- **OPTS** (facoltativo): è una lettera che corrisponde ad una opzione di ordinamento

Vediamo un esempio:

1. Creiamo un file "inventario.txt" con il contenuto:

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

2. Ordiniamo l'inventario per il valore intero presente in ID-[NUMERO]: non c'è bisogno di impostare un separatore perchè **sort** già di default separa i campi tramite uno spazio.

3. ID numerico è contenuto nel primo campo quindi:

```
sort -k 1
```

4. ID numerico parte dalla posizione 4:

```
sort -k 1.4
```

5. ID numerico finisce nella posizione 6 (NB: **ovviamente se la seconda posizione non viene specificata allora si assume il fine riga**):

```
sort -k 1.4,1.6
```

6. Impostiamo un ordinamento:

```
sort -k 1.4,1.6n  
# F[.C][OPTS]
```

Individuazione e ricerca

Individuazione di file dai metadati

Tramite il comando "**find**" individua a partire dai suoi metadati:

```
find [OPZIONI] [PERCORSO] [ESPRESSIONE]  
  
# spiegazione  
# Opzioni: configurano le modalità operative di find  
# Percorso: percorso iniziali di ricerca  
# Espressione: criterio di individuazione su metadati
```

Lanciando il comando da solo effettua una ricerca nella directory attuale, elencando tutti i tipi di file:

```
find
```

Indicando un percorso, stampa tutti i file all'interno del percorso:

```
find /etc
```

Criteri di matching

- Il **criterio di matching** piu **semplice** è il match di un pattern di shell tramite **"-name"**

Nel comando di esempio assume che la directory di ricerca sia **"/etc"** e che il criterio di individuazione sia **"elenca tutti i file che verificano i pattern *.conf"**

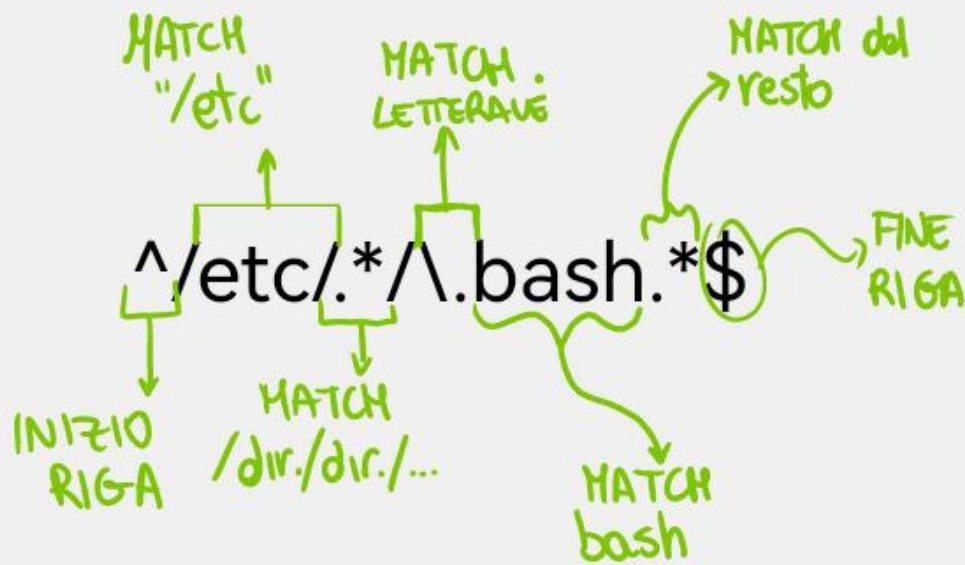
```
find /etc -name *.conf
```

- Esiste anche il criterio tramite **espressioni regolari emacs** (per guardare quali sono: [Parte 7 - File > <u>Espressioni regolari Emacs</u>](#)) e si utilizzando l'opzione **"-regex REGEX"**.

Comando per cercare nella directory /etc tutti i file che iniziano con la stringa .bash

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

Spiegazione del comando



- Esistono le varianti "case insensitive" di **-name** e **-regex** che sono **-iname** e **-iregex**
- E' possibile individuare file in base a valori di altri metadati per esempio:
 - **-atime n**: il file è stato acceduto l'ultima volta n*24ore
 - **-perm mode**: il file ha permessi **mode**
 - **-size n [cwbkMG]**: il file usa n **unità di spazio su disco**
- Per personalizzare la stampa del comando "find" basta utilizzare l'opzione **"-prinf"** e successivamente una

stringa formato che specifica il tipo di stampa voluta (simile al C):

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

In questo caso il comando stampa il nome del file (**%p**) e i suoi permessi in ottale (**%m**)

Invece per stampare anche la dimensione si utilizza (**%s**)

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

- E' possibile anche eseguire un comando durante la ricerca ed è possibile grazie all'opzione **"-exec COMMAND '{}'\;"**

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

In questo esempio, cerca i file che iniziano con il match ".config" in tutto il sistema e ad ogni risultato ne stampa il tipo di file.

Individuazione del file dal contenuto

Tramite il comando **"grep"** individua file a partire dal suo contenuto

```
grep [OPZIONI] pattern [file]
```

```
# Spiegazioni
```

```
# OPZIONI: configurano le modalità operative di grep
```

```
# pattern: espressione di ricerca
```

```
# file: uno o piu file su cui operare
```

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

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

(stampa tutte le stringhe che contengono la stringa root)

Opzioni

- Tramite l'opzione **"-n"** stampa il numero di riga in cui è avvenuto il match:

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

(output in forma "database testuale" separato da un ":")

- Tramite l'opzione **"-H"** stampa il nome del file in cui è avvenuto un match con il pattern:

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

(-H si attiva in automatico quando si concatenano piu file)

- Tramite l'opzione **"-R"** effettua una ricerca ricorsiva nei sottoalberi

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

- Tramite l'opzione **"-i"** effettua una ricerca **case insensitive**

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

- Tramite l'opzione **"--color=yes"** evidenzia i match in colore rosso

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

- E' possibile effettuare una ricerca tramite l'espressioni create tramite l'opzione "-regex" e le espressioni regolari (per vedere tutti i caratteri speciali: [Parte 7 - File > Ricerca tramite espressione regolare](#))

Un esempio di utilizzo di espressioni regolari

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

Allora spieghiamo ogni singolo pezzo del REGEX così da capirci qualcosa:

1. `[[:digit:]]` intercetta un carattere cifra
 2. `{1,3}` - Intercetta fino a tre caratteri cifra consecutivi (visto che gli ip vanno da 1 cifra (1) a 3 cifre (255))
 3. `\.` - Intercetta fino a tre caratteri cifra consecutivi seguiti da un punto
 4. (Blocco) - Considera tutti il match come un blocco
 5. `{3}` - Il blocco precedente (numero seguito da ".") deve essere ripetuto 3 volte
 6. `[[:digit:]]{1,3}` - Segue un carattere cifra fino a 3 cifre
- Tramite l'opzione "-o" stampa esclusivamente la porzione di riga che verifica il match

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

- Tramite l'opzione "-v" stampa tutto tranne quello che corrisponde al matching

Collegamenti

Spiegazione collegamenti

Consistono in una sorta di "**puntatori**" ad un file, tipicamente con un nome alternativo.

Esistono due tipi di collegamenti che adesso elenchiamo

Collegamento fisico (hard link)

Per collegamento fisico si intende la creazione di un nuovo elemento di directory che punta allo stesso contenuto del file originale.

Tramite il comando "**ln**", lanciato senza opzioni crea collegamenti fisici:

```
ln TARGET LINK_NAME
```

```
# Spiegazione
```

```
# TARGET: è il percorso del file originale
```

```
# LINK_NAME: è il nome del collegamento fisico
```

(I METADATI TRA I DUE FILE SARANNO IDENTICI)

Esistono però delle limitazione, per esempio non si possono creare dei collegamenti fisici ad una directory.

Oppure non si possono creare collegamenti ad un file in un altro file system

Infine non si possono creare collegamenti ad un file posseduto ad un altro utente

Collegamento simbolico (soft link)

Il collegamento simbolico è un nuovo file, diverso da quello originale ma con il contenuto identico

Tramite l'opzione "-s" del comando **ln** permette la creazione di un collegamento simbolico:

```
ln -s TARGET LINK_NAME
```

Risoluzione dei collegamenti simbolici

Tramite il comando "**readlink**" è possibile il collegamento simbolico su un file

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

Opzioni

- Tramite l'opzione **-f** consente di risolvere collegamenti in maniera ricorsi, cioè si ottiene il file puntato dalla catena di collegamenti simbolici

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

```
# output  
/bin/nano
```

Archivi

Creazione di un archivio tramite il comando tar

Il primo comando che vedremo per la gestione degli archivi è "**tar**" (tarball) e permette di creare, eliminare e elencare i diversi archivi.

Opzioni

- Tramite la fusione di due opzioni che sono "-f" che serve per la selezione/riferimento del file e "-c" si crea un archivio, accettando come argomento un elenco di file e/o directory FD1, FD2, FDN:

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

- Tramite l'opzione "-t" e "-f" è possibile vedere l'elenco di file/directory all'interno di un archivio

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

- Tramite l'opzione "-x" si estrae un archivio:

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

- Tramite l'opzione "-v" abilita la modalità verbosa, cioè stampa i file e directory manipolati durante l'operazione:

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

- Tramite l'opzione "-C DIRECTORY" si specifica nella quale tar preleva i file o dove deve scriverli.

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

NB: ha effetto su solo sulle opzioni ad esse successive;
molto importante la posizione di "-C DIRECTORY"

Opzioni di compressione

- Tramite queste diverse opzioni è possibile la compressione in più formati:
 - Opzione -z → Compressione GZIP (.gz)
 - Opzione -j → Compressione BZIP2 (.bz2)
 - Opzione -J → Compressione XZ (.xz)

Dal meno al più compresso

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

Dettaglio di tar

E' possibile avviare le opzioni senza i trattini per colpa delle diverse diaspore uscite con UNIX.

Creazione di un archivio tramite il comando 7z

Tramite il comando "7z" si possono gestire degli archivi e rappresenta un'alternativa moderna di tar (visto anche la possibilità di gestire dei formati molto più recenti):

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

Opzioni

- Tramite il comando "a" è possibile creare un archivio, con argomenti un elenco di file/directory FD1, FD2, ... FDN

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

- Tramite il comando "l" è possibile elencare il contenuto di una directory:

```
7z l ARCHIVE_NAME
```

```
# esempio
```

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

- Tramite il comando "-x" è possibile estrarre un archivio:

```
7z x ARCHIVE_NAME
```

```
# esempio
```

```
7z x config.7z
```

- Tramite l'opzione "-m" abilita delle funzione aggiuntive. Ad esempio la possibilità di cifrare un archivio con la chiave password ecco un esempio:

```
7z -mhe=on -password PASSWORD ARCHIVE_NAME FD1
```

Bash e file

Operatori unari su file

Anche bash mette a disposizione tanti operatori unari per la verifica di proprietà su file. Ecco l'elenco dei principali:

1. **-d FILE VERO** se FILE esiste ed è una directory
2. **-e FILE VERO** se FILE esiste
3. **-f FILE VERO** se FILE esiste ed è un file regolare
4. **-h FILE VERO** se FILE esiste ed è un link simbolico
5. **-r FILE VERO** se FILE esiste ed è leggibile
6. **-w FILE VERO** se FILE esiste ed è scrivibile
7. **-x FILE VERO** se FILE esiste ed è eseguibile

Ad esempio, per scoprire se un file è leggibile o meno, si esegue un test di leggibilità e si stampa il codice d'uscita per esempio:

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

# output
0 # cioè leggibile
```

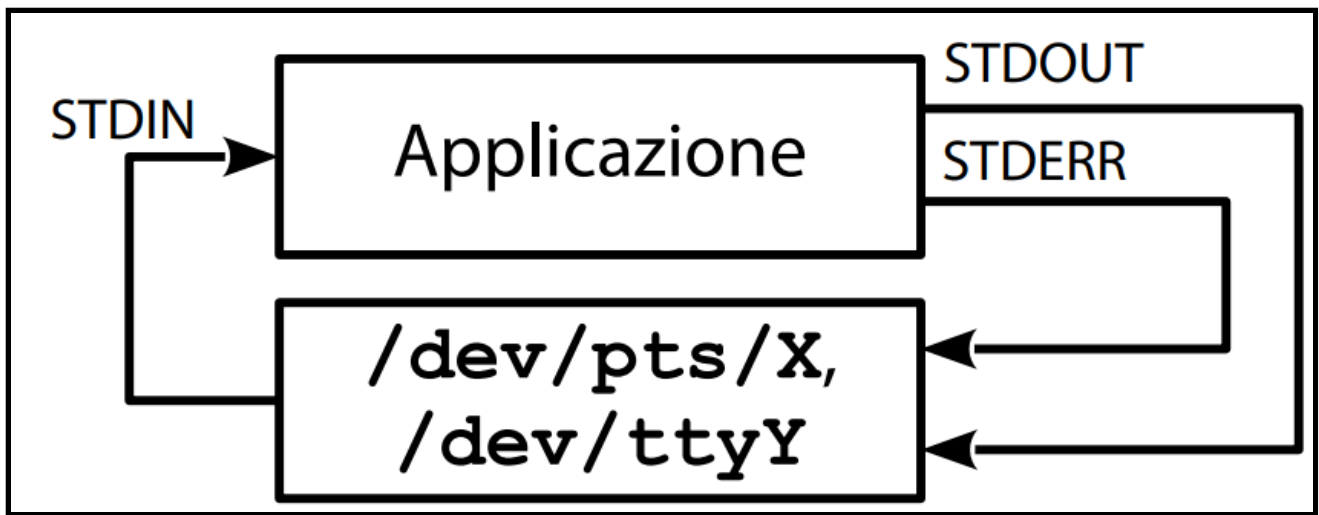
Canali di I/O

Ogni applicazione in esecuzione ha almeno tre **Canali di I/O**:

1. **STDIN**: Legge gli input dall'utente
2. **STDOUT**: Scrive gli output
3. **STDERR**: Scrive i messaggi d'errore



I tre canali sono associati al file speciale del dispositivo di terminale connesso all'istanza di BASH in esecuzione.



E i tre file hanno collegamenti simbolici ai canali STDIN, STDOUT E STDERR dell'applicazione attualmente in esecuzione sul terminale corrente:

```
/dev/stdin  
/dev/stdout  
/dev/stderr
```

Il descrittore di file

Un canale è accessibile tramite il suo descrittore di file, che consiste in un numero intero trappresentante un indice ad un file aperto da una applicazione:

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

Operatore "<"

L'operatore di redirezione `0< FILENAME` associa il canale STDIN ad un file. (**ESISTE ANCHE LA SINTASSI RIDOTTA `< FILENAME`**):

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

Operatore ">"

L'operatore di redirectione 1> FILENAME associa il canale di STDOUT ad un file. (**ESISTE ANCHE LA SINTASSI RIDOTTA > FILENAME**):

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

L'operatore di redirectione 2> FILENAME associa il canale di STDERR ad un file

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

Operatore ">>"

L'operatore di redirectione >> FILENAME appende l'output di un canale ad un file:

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

Esempio:

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

Operatore ">&"

L'operatore di redirectione `N>&M` copia il descrittore di file `N` nel descrittore di file `M`. D'ora in poi, il canale puntato da `N` scrive sullo stesso file puntato da :

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

Esempio:

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

Gestione di file

Tramite il builtin **exec**, utilizzato con **opportuni operatori di redirectione, apre, redirectione, chiude file su descrittori arbitrari**. E' possibile usare fino a 1024 descrittori diversi per ogni istanza di shell.

Esempio, per aprire il file `/etc/passwd` ed associarlo al descrittore di file `3`, si esegue il comando:

```
exec 3< /etc/passwd
```

Invece ora per leggerlo utilizziamo l'opzione `"-u"` di **read** (che specifica il descrittore di file da cui leggere):

```
# comandi
read -u 3 line
echo &line

# output
root:x:0:0::/root:/bin/bash
```

E' possibile effettuare lo stesso procedimento però in scrittura:

```
exec 4> my-output.txt
```

```
# Descrittore numero 4
```

Si scrive il contenuto della variabile **line** nel file appena aperto

```
echo $line >&4
```

Si stampa il file:

```
# comandi
```

```
cat my-output.txt
```

```
# output
```

```
root:x:0:0::/root:/bin/bash
```

NB: RICORDARSI DI CHIUDERE I FILE! (Tramite "-" in fondo al comando exec):

```
exec 3<&-
```

```
exec 4>&-
```

Filtri UNIX

I comandi UNIX per l'elaborazione dei testi sono concepiti come veri e propri **filtri**, cioè una applicazione che trasforma il flusso di dati in ingresso (**input stream**) in un flusso di dati in uscita (**output stream**).

Il flusso d'entrata è letto dal canale STDIN, invece il flusso d'uscita è letto dal canale STDOUT, gli errori sul canale STDERR

Modalità standalone

Un singolo filtro gestisce il tutto da solo, tipicamente in modalità interattiva, modalità per principianti.

Modalità combinata

La soluzione più semplice per combinare filtri consiste nel memorizzare manualmente l'output di un comando in un file temporaneo, che sarà usato dal comando successivo:

```
COMANDO1 > tmp.txt  
COMANDO2 < tmp.txt
```

Questo metodo richiede della gestione extra da parte dell'utente.

- Un esempio di utilizzo di un filtro: si consideri la trasformazione su `/etc/passwd`: stampa delle colonne 1 e 7 e stampa delle prime 10 righe

Viene condotta in due passi questa trasformazione tramite i comandi **cut** e **head**.

Stampa le colonne 1 e 7:

```
cut -d ":" -f 1,7 /etc/passwd > tmp.txt
```

Stampa le prime 10 righe:

```
head tmp.txt
```

Autenticazione

Classificazione utenti

Utenti normali

Tutti gli **utenti normali** possono **eseguire una shell interattiva al loro login**: `"/bin/bash"` che può essere cambiata, e gli utenti in `"home/username"` **hanno una home directory che può essere cambiata**

NON possono modificare configurazione sulle risorse hw/sw.

Utenti superutenti

Un utente in grado di fare tutto, di fatto un **amministratore** (o **superutente**) con identificatore utente: 0 e il suo nome è `"root"`

Anche loro possono scegliere una shell interattiva al loro login e hanno una home directory `"/root"` che può essere cambiata

Può alterare le configurazioni di risorse hw/sw

File `/etc/passwd`

Contiene l'elenco degli utenti noti al SO, con un elenco di record (riga per riga) separati da `":"` e questo file è: leggibile da tutti gli utenti e modificabile solo da **root**

E i campi sono:

1. Nome del login
2. Password cifrata
3. Identificatore utente
4. Identificatore del gruppo di lavoro del file
5. Nome e cognomi veri
6. Home directory
7. Interprete dei comandi usato

Il file /etc/gshadow

Contiene tutte le **informazione sulle password associate a ciascun gruppo**, un elenco di record (uno per riga) e i campi sono separati dal carattere ":" ed è leggibile e modificabile solo dall'utente **root**.

I campi all'interno del file sono:

1. Nome del gruppo
2. Password cifrata
3. Utenti amministratori del gruppo
4. Utenti membri del gruppo

Gestione utente

Creazione di un utente in Debian

Tramite il comando "adduser" è possibile creare un nuovo utente (eseguita da amministratore):

```
adduser nome_di_login
```

NB: Il comando è **interattivo** cioè il terminale comunica con l'utente.

Il nuovo utente avrà **a disposizione una home directory** "/home/prova", un gruppo primario di lavoro **prova** e un interprete di default (/bin/bash) con una **configurazione iniziale copiata** dalla directory /etc/skel (skeleton)

Rimozione di un utente in Debian

Tramite il comando "deluser" è possibile cancellare un utente:

```
deluser nome_di_login
```

NB: Questo comando non è **interattivo**

Il risultato del comando è la rimozione completa dell'utente, di tutto, dal nome di login, dalla cancellazione del gruppo. (--remove-home come **opzione prima di fare deluser per cancellare anche la sua home directory**)

Creazione di un utente in GNU/LINUX

Tramite il comando "**useradd**" è possibile creare un utente in ambienti GNU/Linux

```
useradd prova
```

NB: Non è **interattivo** e non crea neanche la home directory

Per creare una home directory

```
useradd -m prova
```

Rimozione di un utente in GNU/Linux

Tramite il comando "**userdel**" è possibile cancellare un utente in ambienti GNU/Linux, ma non cancellare la home directory, per farlo basta aggiungere l'opzione "**-r**":

```
userdel -r prova
```

Configurazione piu completa di utenti

Tramite l'opzione -G è possibile impostare un gruppo di lavoro secondario:

```
adduser --ingroup video prova # comando debian  
useradd -G video -m prova # comando altre distribuzioni
```

Modifica di un utente già esistente

Tramite il "**usermod**" permette di modificare le proprietà e le risorse di un utente già esistente

```
usermod [opzioni] nome_di_login
```

Opzioni

- "-l": cambia il nome di login
- "-u": cambia l'identificatore utente
- "-d": specifica la nuova home in /etc/passwd
- "-m": spostare il contenuto della vecchia directory nella nuova

Gestione gruppi

Creazione/rimozione gruppi in Debian

Tramite il comando "**addgroup**" si crea un nuovo gruppo:

```
addgroup nome_gruppo
```

Il risultato del comando sarà quella di una creazione di un gruppo nuovo di zecca.

Tramite il comando "**delgroup**" si rimuove un gruppo passato come parametro:

```
delgroup nome_gruppo
```

NB: Non è interattivo

Il risultato del comando è la rimozione completa del gruppo

Creazione/Rimozione gruppi in GNU/LINUX

Causa diaspore varie, al di fuori di debian si utilizzano altri comandi per la creazione e la cancellazione di gruppi.

Tramite il comando "**groupadd**" si crea un gruppo e tramite "**groupdel**" si rimuovono, uguali dal punto di vista concettuale

Modifiche di un gruppo esistente

Tramite il comando "**groupmod**" permette di modificare le proprietà e le risorse di un gruppo esistente:

```
groupmod [opzioni] nome_gruppo
```

Opzioni

- Tramite l'opzione "-n" è possibile modificare il nome del gruppo
- Tramite l'opzione "-g" modifichi l'identificatore del gruppo

Analisi dei gruppi di un utente

Tramite il comando "**groups**" mostra i gruppi di appartenenza di un utente:

```
groups nome_di_login
```

(avviato senza argomenti stampa i gruppi esistenti)

Modifiche dei gruppi di un utente

Tramite il comando **usermod** è usato per modificare i gruppi di lavoro primario e secondario:

- L'opzione "-g nome_del gruppo" specifica il nuovo gruppo di lavoro primario
- L'opzione "-G ng_1, ng_2, ng_3, ..." specifica il nuovo insieme completo dei gruppi secondari

Ad esempio vogliamo impostare all'utente "prova" come gruppo primario: root e come gruppi secondari: "disk cdrom floppy audio"

I comandi richiesti (servono credenziali da root):

```
usermod -g root prova  
usermod -G disk cdrom floppy audio prova
```

Per evitare di cancellare tutti i gruppi secondari precedenti, basta utilizzare l'opzione "-a" che funziona da "append", così da non cancellare tutti i gruppi precedenti.

Comando alternativo

Tramite il comando **gpasswd** è possibile modificare gruppi di lavoro per gli utenti selezionati

Per aggiungere un utente ad un gruppo si utilizza l'opzione "-a":

```
gpasswd -a nome_di_login gruppo
```

Per rimuovere un utente da un gruppo si utilizza l'opzione "-d":

```
gpasswd -d nome_di_login gruppo
```

Ad esempio per aggiungere il gruppo "lp" all'utente prova:

```
gpasswd -a prova lp
```

Per toglierlo invece basta cambiare "-a" con "-d".

NB: Le modifiche non saranno visibili fino al prossimo login

Gestione permessi

Modificare l'utente creatore

Tramite il comando "**chown**" modifica l'utente creatore di un file e/o di una directory:

```
chown [OPZIONI] nome_di_login file
```

Modifica del gruppo del file creatore

Tramite il comando "**chgrp**" si modifica il gruppo creatore di un file e/o di una directory:

```
chgrp [OPZIONI] nome_del_gruppo file
```

Modifica di utente creatore e gruppo

E' possibile effettuare simultaneamente la modifica sia dell'utente creatore che del gruppo tramite questa scorciatoia:

```
chown nome_di_login:gruppo file
```

Ad esempio, per modificare tutti i due i parametri contemporaneamente le modifiche sul file "file.txt":

- utente creatore: studente
- gruppo del file: studente

è possibile eseguire il solo comando **chown**:

```
chown studente:studente /home/studente/file.txt
```

Modifica dei permessi del file

Tramite il comando "**chmod**" è possibile modificare i permessi dei file:

```
chmod [OPZIONI] permessi file
```

I permessi possono essere rappresentati in due modi:

- Rappresentazione testuale:

I permessi da applicare ad una directory o un file sono rappresentati tramite una o più **stringhe di permessi** separati da una virgola (str1, str2, ..., strn) e ciascuna riga di permessi ha il formato seguente.

```
insieme_di_utenti±insieme_di_permessi
```

Il simbolo \pm indica la presenza di un aggiunta o di una rimozione.

L'insieme di utenti è una stringa composta dai caratteri:

1. **u** → permessi si applicano all'utente creatore
2. **g** → permessi si applicano al gruppo del file

3. **o** → permessi si applicano ai restanti utenti

4. **a** → **permessi si applicano a tutti gli utenti**

Ad esempio:

ugo → tutti gli utenti del sistema

a → tutti gli utenti del sistema

go → solo gruppo e il resto del mondo (xD)

L'insieme dei permessi è una stringa composta dai caratteri:

1. **r** → si applica il permesso di lettura

2. **w** → si applica il permesso di scrittura

3. **x** → si applica il permesso di esecuzione (file) o di accesso dir.

4. **s** → si applica il bit SETUID/SETGID

Ad esempio:

rwX → lettura, scrittura, esecuzione

rw → lettura, scrittura

rx → lettura, esecuzione

rws → lettura, scrittura, esecuzione SETUID/SETGID

Un esempio di esecuzione del comando **chmod**:

Vogliamo assegnare al file di testo **/home/studente/file.txt** il seguente insieme di permessi: **rw-rw-r--**

Dobbiamo costruire la stringa di permessi:

- Per l'utente creatore e per il gruppo dobbiamo aggiungere il permesso di scrittura e di lettura, quindi selezioniamo il gruppo "ug" e i permessi "rw",

- Per il resto degli utenti va fatto solo il permesso della scrittura quindi prendiamo i gruppi e i permessi giusti "o" e "r"

Formiamo il comando

```
chmod ug+rw,o+r /home/studente/file.txt
```

- Rappresentazione in ottale

I permessi da **applicare ad una directory o ad un file sono rappresentati tramite un numero in base 8** (ottale).

Il numero può essere lungo da una a quattro cifre (in caso sia più corta di 4, si aggiungono degli zeri)

Ad esempio: 4 → 0004, 755 → 0755

Consideriamo un insieme di permessi (0755)

- **Seconda cifra (7)**: Insieme di permessi per l'utente creatore
- **Terza cifra (5)**: Insieme di permessi per gli utenti appartenenti al gruppo del file
- **Quarta cifra (5)**: Insieme di permessi per gli altri utenti

L'insieme dei permessi è un numero intero lungo 4 bit (da 0 a 7) ottenibile sommando i numeri (0, 4, 2, 1):

0 → nessun permesso

4 → Si applica il permesso di lettura

2 → Si applica il permesso di scrittura

1 → Si applica il permesso di esecuzione/ingresso

Quindi di conseguenza:

1. 7 → Tutti i permessi
2. 6 → Lettura e scrittura
3. 5 → Lettura ed esecuzione/ingresso
4. 4 → Lettura
5. 3 → Scrittura, esecuzione/ingresso
6. 2 → Scrittura
7. 1 → Esecuzione/ingresso

8. 0 → nulla

Per quanto riguarda la prima cifra invece stabilisce ulteriori permessi (SETUID, SETGID):

- **Prima cifra = 4** → E' impostato il bit SETUID
 - **Prima cifra = 2** → E' impostato il bit SETGID
 - **Prima cifra = 1** → E' impostato lo sticky bit
 - **Prima cifra = 0** → Nulla
-

Processi

Individuazione dei processi

Identificatore di un processo

Uno dei campi contenuti nel descrittore di processo è **l'identificatore di processo (Process Identifier, PID)**

Uno dei campi contenuti nel descrittore di thread è **l'identificatore di thread (Thread Identifier, TID)**.

Il PID/TID è un numero intero non negativo assegnato dal nucleo all'istante di creazione di un nuovo processo

Tramite il comando esterno "**pidof**" stampa i PID di tutti i processi attivi a partire da un **nome esatto**

```
pidof bash
```

Una limitazione di pidof che si può stampare il PID solo di nomi esatti e non di nomi parziali.

Tramite il comando esterno "**pgrep**" stampa tutti i PID di tutti i processi attivi a partire da una espressione regolare

Si provi a stampare i PID di tutti i processi il cui nome inizia con la stringa "ba"

```
pgrep '^ba.*$'
```

Opzioni pgrep

1. "pgrep -n bash": stampa il PID del processo più recente con il nome bash
2. "pgrep -o bash": stampa il PID del processo meno recente con il nome bash
3. "pgrep -l bash": stampa il PID e il nome dei processi contenenti la stringa "bash" nel nome.

Riproduzione

Creazione di processi

Durante la propria esecuzione, un processo può creare altri processi. Il meccanismo di creazione è una vera e propria **clonazione (Forking)**:

Processo **clonante**: processo **padre**

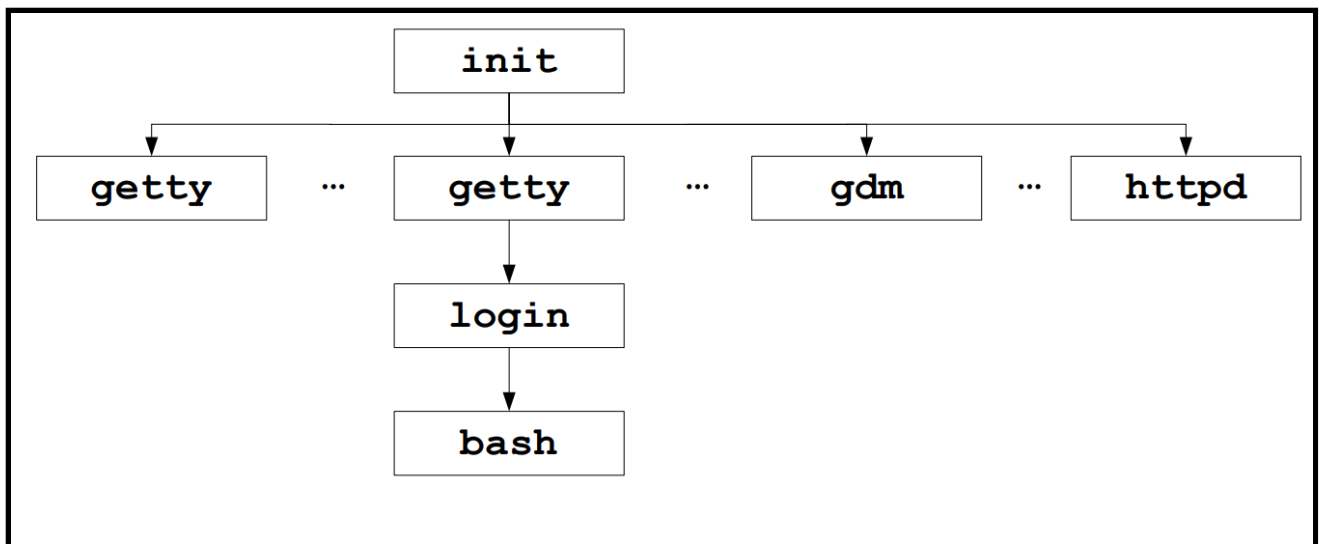
Processo **clonato**: processo ****figlio**

Un esempio banale di una creazione di gerarchie è la creazione di tante finestre bash.

Albero dei processi

Organizzazione processi in SO UNIX

Nei SO UNIX i processi sono organizzati in un albero detto **albero dei processi (process tree)**



Visualizzazione albero dei processi

Tramite il comando esterno "**pstree**" stampa una rappresentazione compatta dell'albero dei processi:

```
pstree | less -Mr
```

NB: Illeggibile in queste condizioni

Opzioni

- Tramite l'opzione "**-a**" stampa i nome dei processi in esecuzione ed i relativi argomenti

```
pstree -a
```

- Tramite l'opzione "**-c**" stampa una rappresentazione dell'albero più esplicita
- Tramite l'opzione "**-p**" mostra tutti i pid dei processi

- Tramite l'opzione **"-H" PID**: evidenzia in neretto il ramo dell'albero dal processo **init** al processo identificato da **PID**

Pipe

La **pipe (tubazione)** è un meccanismo di comunicazione unidirezionale (half-duplex) e sequeziale fra un processo scrittore e un processore lettore.

In bash, l'operatore **"|"** implementa il meccanismo delle pipe nella forma seguente:

```
COMANDO1 | COMANDO2 | ... | COMANDON
```

Il comando composto ora visto ha la semantica seguente:

- COMANDO1 legge lo STDIN da terminale o da file
- Lo STDOUT di COMANDO_i punta allo STDIN di comando COMANDO_{i+1}. Pertanto, COMANDO_{i+1} legge in input l'output di COMANDO_i.
- Di conseguenza il COMANDON legge lo STDIN da comandon-1 e produce un output su terminale o su file

Named pipe

Una **named pipe** (O FIFO) è quasi identica ad una pipe, con l'unica differenza è che gli STDIN E STDOUT dei processi si riferiscono ad un file speciale su disco.

Il file speciale ha lunghezza zero ed una propria struttura di metadati (inode)

Il file speciale è riconoscibile dal **primo carattere (p)** nell'output del comando `ls -l`

Il comando **mkfifo** crea un file speciale per una named pipe:

```
mkfifo FILENAME
```

Il file è marcato **p** ed ha lunghezza pari a zero

Job Control

Il **job control** è un insieme di comandi e meccanismi che consente di gestire l'esecuzione simultanea di più comandi complessi sullo stesso terminale.

Il job control fornisce strumenti di gestione del gruppo di processi associati ad un comando semplice o complesso:

- Identificazione
- Esecuzione "Agganciata" ad un terminale
- Esecuzione "sganciata" dal terminale
- Sospensione, ripristino
- Interruzione, terminazione

Il generico comando lanciabile dall'interprete dei comandi BASH prende il nome di pipeline e ha la forma seguente:

```
C1 [ [ |||&] C2 ... [ |||&] CN]
```

Si possono eseguire uno o più comandi separati dagli operatori
| o |&

Gruppo di processi

Un **gruppo di processi** (process group o job) è l'insieme dei processi che partecipano ad una pipe.

L'ultimo processo della pipeline è detto **process group leader**; il suo stato di uscita è ritornato a BASH ed è rappresentativo dell'intero pipeline

Il comando jobs

Il comando interno **jobs** permette di elencare i job lanciati:

```
jobs
```

E' possibile eseguire delle pipeline sullo sfondo appendendo l'operatore "&" al termine di essa

La pipeline seguente esegue sullo sfondo:

```
sleep 7200 &  
  
# output  
[2] 2231  
# job id con process group ID
```

Stampando ancora l'elenco di tutti i processi notiamo come esce un altro risultato in esecuzione è marcato con il carattere "-" che identifica il **job precedente**, un metodo per fornire all'utente due scorciatoie (+ e -) per alternare rapidamente in primo piano i due job più "recenti".

Rimettere in primo piano un job

Tramite il comando **fg** si può rimettere il comando in primo piano

Un job può essere rappresentato in diversi modi:

%JOB_ID

%COMANDO (o sottostringa di COMANDO)

%+ o %

%-

Per ripristinare il job 1 si può eseguire ad esempio:

```
fg %1
```

Rimettere sullo sfondo un job

Tramite il comando "**bg**" (BackGround) si può rimettere il comando in background:

```
bg %1
```

il job può essere rappresentato con i stessi modi di sopra

Bash ed esecuzione di processi

Sostituzione di comando

L'operatore `$(COMMAND)` introduce una espansione di parametro nota come il nome di **sostituzione di comando** (**command substitution**)

Viene eseguito il comando `COMMAND` e il suo output viene sostituito dall'espressione `$(COMMAND)`

Ad esempio per memorizzare in una variabile intera a il numero di linee di `/etc/passwd`:

```
declare -i a  
a=$(wc -l /etc/passwd | cut -f1 -d" ")
```

Sostituzione di processo

Gli operandi `<(COMMAND)` e `>(COMMAND)` introducono una espansione di parametro nota come il nome di **sostituzione di processo** (**process substitution**)

- Se si utilizza **<(COMMAND)**, l'output del comando viene fatto vedere come un file al comando che invoca **<()**

Ad esempio, per eseguire la seguente trasformazione su `/etc/passwd`:

- Stampa una colonna di interi crescenti, la prima e l'ultima colonna di `/etc/passwd`
- limita l'output alle prime dieci righe di `/etc/passwd`

Si può eseguire il comando seguente:

```
paste <(seq 1 10) <(cut f1 -d":" /etc/passwd | head) <(cut -f7 -d":" /etc/passwd | head)
```

(Alternativa e quella di salvare i comandi intermedi in file tramite `> out.txt`)

- Se si usa **>(COMMAND)**, l'output scritto su un file di nome `/dev/fd/N` dal comando che usa **>()** viene fatto vedere come input (STDIN) al comando **COMMAND**.

Ad esempio, per far elaborare al volo da più programmi il file `/etc/passwd` si può eseguire il comando:

```
tee < /etc/passwd  
    >(COMANDO1)  
    >(COMANDO2)
```

Fork bomb

Cosa si intende per Fork bomb?

Una **fork bomb** è un processo che provoca la creazione di un numero sproposito di figli, è un meccanismo rozzo ma efficace

di negazione del servizio (Denial of Service, DOS)

Esempio di fork bomb:

```
:(){ :|:& };
```

```
# Spiegazione:
```

```
# :() si definisce una funzione bash
```

```
# {} indica il corpo della funzione
```

```
# :|: richiama due comandi in pipe quindi crea un processo padre con due figli
```

```
# & Esecuzione in backgroup della pipe ha un effetto collaterale, se termina il padre i suoi processi figli non terminano quindi continuano all'infinito a creare altri processi
```

```
# Non basta terminare il processo padre per terminare la fork bomb. È necessario terminare tutti i processi. Tuttavia, il nucleo è più veloce a creare processi che a terminarli..
```

```
# ; il separatore di bash
```

```
# : avvia le danze -> invoca la funzione
```

Elevazione privilegi

Elevazione con su

Visione degli identificatori

Tramite il comando "**id**" stampa gli identificatori utente, di gruppo primario e dei gruppi secondari:

```
id [OPZIONI] ... [UTENTE]
```

Esecuzione di una shell

Tramite il comando "**su**", così da solo attiva "una maschera" che fa diventare l'utente in **root** ma usando le variabili d'ambiente dell'utente che si è elevato.

Tramite il comando "**su -**" invece si diventa proprio nell'ambiente di *root* usando le sue variabili d'ambiente e tutto.

E' possibile anche diventare un altro utente specifico lanciando una sua shell:

```
su - nome_di_login
```

Esecuzione di comandi generici

E' possibile anche lanciare dei comandi specifici con la shell di un altro utente tramite l'opzione "**-c**":

```
su -c [COMANDO] - nome_di_login
```

Se il comando è più complesso è necessario usare le virgolette:

```
su -c "COMANDO COMPLESSO" - nome_di_login
```

Elevazione con sudo

Tramite il comando esterno "**sudo**" esegue un **comando come un altro utente e/o gruppo**

Proprio come il comando **su**, con la differenza che di default **sudo** chiede la password **dell'utente di partenza**, non di quello

di arrivo (quindi senza sapere la password dell'utente che vogliamo utilizzare)

Il file /etc/sudoers

Prima di utilizzare sudo, bisogna configurarlo e per farlo bisogna farlo nel file "/etc/sudoers"

Tramite il comando "**visudo**" (eseguito come root) apre un editor per configurare il file sudoers.

NB: Il comando **visudo** utilizza la variabile d'ambiente **\$EDITOR** per capire quale editor eseguire.

Per cambiarlo:

```
EDITOR=vim visudo # specifica l'editor vim
```

E' composto da delle proprietà:

- **env_reset**: i comandi lanciati con sudo eseguono in un ambiente ristretto comprendente poche e selezionate variabili di ambiente
- **mail_badpass**: invia un email all'indirizzo di posta dell'amministratore in caso di tentativi di password errata.
- **secure_path**: i comandi lanciati con **sudo** ricevono una variabile d'ambiente **PATH** con il valore specificato nell'argomento

Specifica dei privilegi utente


```
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

Queste sono le specifiche di privilegi per utenti e gruppi, la prima è caratterizzata da:

<who> <where>=<as whom> <what>

<who>: chi ha il permesso di eseguire, in questo caso root

<where>: da quali host il permesso di eseguire, in questo caso tutti gli host (nessun limite di ip) ALL

<as whom>: chi può diventare prima di eseguire, in questo caso può trasformarsi in un qualunque utente e in qualunque gruppo (ALL:ALL)

<what>: cosa può eseguire, in questo caso tutti i tipi di comandi

La seconda specifica è molto simile alla prima specifica tranne per la presenza del carattere % prima del primo campo.

Il carattere corrisponde all'interpretazione della stringa come nome del gruppo.

Lettura dei proprio privilegi

Tramite il comando e l'opzione "**sudo -l**" è possibile visualizzare che cosa può fare l'utente attuale

```
sudo -l
```

Tramite il comando e l'opzione "**sudo -ll**" produce un output più prolisso

```
sudo -ll
```

Tramite il comando e l'opzione "**sudo -llU**" specificando il nome di login dell'utente, si specificano i privilegi di lancio:

```
sudo -llU docente
```

Aggiunta al gruppo sudo

Tramite il comando "**gpasswd**" ci si aggiunge la gruppo **sudo** (mettendo in **append** il gruppo (non messo cancella tutti i gruppi nella quale è presente l'utente specificato) **tramite** l'opzione "**-a**"):

```
gpasswd -a docente sudo
```

Esecuzione di un comando

Tramite l'opzione "**-u**" è possibile specificare con quale utente eseguire il comando:

```
sudo -u docente id
```

Tramite l'opzione "**-g**" è possibile specificare anche il gruppo con cui eseguire il comando:

```
sudo -u docente -g root id
```

Esecuzione globale per utente/gruppo senza richiedere una password

⚠ NB: NON FARLO MAI ⚠

Andando a modificare sempre il file sudoers è possibile inserire delle etichette, in tutte e due le sezioni sia per i gruppi che per

gli utenti

In questo esempio andiamo a vedere come togliere la richiesta di password in ogni occasione:

```
%sudo ALL=(ALL:ALL) NOPASSWD: ALL
```

Esecuzione comandi specifici

E' possibile inserire al posto di ALL, un elenco di comandi specifici che volete siano eseguiti con delle caratteristiche:

```
%sudo ALL=(ALL:ALL) /path/to/cmd1,/path/to/cmd2,...
```

⚠ Va sempre specificato il percorso assoluto del comando ⚠

⚠ ESEMPIO D'ESAME ⚠

Per il comando **slabtop** mostra le allocazioni di memoria del kernel.

Ovviamente per motivi di sicurezza è permessa solo all'utente root

Se volete che chi è nel gruppo sudo possa avviarlo senza password si può fare:

```
# REMINDER: SEMPRE NEL FILE SUDOERS  
%sudo ALL=(ALL:ALL) NOPASSWD: /usr/bin/slabtop
```

Oppure se volete fare in modo che un utente possa avviare il comando precedente possiamo fare in questo modo:

```
studente ALL=(ALL:ALL) /usr/bin/slabtop
```

Elevazione con SETUID/SETGID

Fa eseguire i comandi in **modalità root** se il SETUID/SETGID è impostato

chmod u+s: impostazione bit SETUID.

chmod g+s: impostazione bit SETGID.

chmod u-s: rimozione bit SETUID.

chmod g-s: rimozione bit SETGID

Elevato con Capability

Lettura di capability

Tramite il comando "**getcap**" mostra le capability di un file e il suo uso è molto semplice:

```
getcap [OPZIONI] file...
```

Ad esempio

```
getcap /bin/ls
```

⚠ Offre piene funzionalità solo quando è eseguito da **root** ⚠

Per scoprire tutti le capability si utilizza la wildcard "*":

```
getcap /bin/*
```

Le capability **mostrate dal comando getcap** sono rappresentate da un insieme di **clausole separate dal carattere ","** (virgola).

```
comando = clausola1, clausola2,...,clausolaN
```

L'azione sulle capability è uno dei tre caratteri seguenti

- "+" → aggiunge le capability
- "-" → rimuove le capability
- "=" → rimuove le capability considerate dagli insiemi effective, permitted ed inheritable, prima di impostarle.

Gli insiemi delle capability sono uno o più dei seguenti insiemi:

- e → insieme delle capability effettive (effective)
- p → insieme delle capability permesse (permitted)
- i → insieme delle capability ereditate (inherited)

Impostare le capability

Tramite il comando "**setcap**" è possibile impostare le capability di un file:

```
setcap [OPZIONI] CAPABILITY FILE
```

Ad esempio:

```
setcap cap_net_raw+ep /path/to/ping  
  
# ping è un comando con le capability cap_net_raw
```

Gestione Software Debian

Gestione repository

Il file /etc/apt/sources.list

Contiene la definizione dei repository usabili dal SO Debian installato.

E' un elenco di righe, ciascuna definente uno specifico repository

Il formato generico è:

```
deb[-src] <URI> <distib.> [componenti]...
```

Il primo campo è una stringa che può assumere uno dei valori seguenti. deb: definizione di un repository per pacchetti binari. deb-src: definizione di un repository per pacchetti sorgente

Sincronizzazione dei repository

Tramite il comando "**apt update**" aggiorna i repository di Debian, è una **operazione propedeutica**.

▲ da fare ogni volta che si scarica un pacchetto ▲

Installazione e rimozione dei pacchetti binari

Installare un pacchetto

Tramite il comando "**apt install**" (eseguito da amministratore) installa il pacchetto P1, P2, ..., PN ed esegue queste operazioni:

1. risoluzione e stampa delle dipendenze.
2. recupero dei pacchetti da sorgenti remote.
3. pre-configurazione (script).
4. spaccettamento.
5. configurazione.
6. post-configurazione (script).

Esempio: ricordarsi di installare le chiavi (firme digitali)

```
apt install debian-keyring
```

Questo pacchetto software installa le chiavi pubbliche GPG degli sviluppatori Debian con uno script

Esempio scaricare il pacchetto **hello**:

```
apt install hello
```

Rimozione di un pacchetto binario

Tramite il comando "**apt remove**" (amministratore) rimuove i pacchetti

Operazioni effettuate dal comando:

1. stampa l'elenco dei pacchetti che verranno rimossi (incluse le dipendenze non più utilizzate);
2. stampa lo spazio su disco liberato dal processo di rimozione dei pacchetti;
3. rimuove tutti i file contenuti in tutti i pacchetti.

Per cancellare il pacchetto scaricato prima "hello" basta fare:

```
apt remove hello
```

⚠ NON CANCELLA COMPLETAMENTE IL PACCHETTO ⚠

Per farlo si deve fare:

```
apt purge hello
```

Così la configurazione è stata interamente cancellata.

Posizione della cache dei pacchetti scaricati

Per scovare dove **sono i pacchetti scaricati** vengono messe direttamente nella directory "cache":

```
ls -l /var/cache/apt/archives
```

Quanto spazio occupano?

Per scoprire quanto pesano i pacchetti:

```
du -hs /var/cache/apt/archives
```

```
# -s total consume  
# -h human readable
```

Cancellazione della cache dei pacchetti

Il comando "**apt clean**" rimuove i file *.deb dalla directory cache del sistema **APT: /var/cache/apt/archives**

```
apt clean
```

Aggiornamento distribuzione

Per aggiornare ad una nuova distribuzione (dopo aver modificato opportunamente il file di configurazione sources.list), si utilizza il seguente comando (da amministratore):

```
apt full-upgrade
```

Per ragioni di compatibilità è accettata anche il sottocomando precedentemente in uso:


```
apt dist-upgrade
```

Perché dist-upgrade e non un semplice upgrade?

Nel cambio di distribuzione possono subentrare modifiche nelle dipendenze fra pacchetti (alcune librerie spariscono, ne entrano altre, ...):

Dist-upgrade gestisce tali **modifiche** (rimuove le vecchie dipendenze ed installa le nuove).

Upgrade non gestisce tali **modifiche** (prova ad avanzare tutti i pacchetti alla nuova versione).

Ricerca e individuazione pacchetti

Ricerca sui pacchetti binari

Tramite il comando "**apt search REGEX1 REGEX2**" stampa diverse informazioni dei pacchetti trovati tramite il regex:

1. Nome
2. Descrizione breve

Esempio cerchiamo tutti i pacchetti il cui nome esatto inizia con la stringa **tmux**:

```
apt search "^tmux.*$"
```

Opzioni

Tramite l'opzione "**--name-only**" esclude la ricerca del regex sulla descrizione breve, ma si concentra solo sul nome.

```
apt search --name-only "^tmux.*$"
```

Convenzione nei nomi di pacchetti

Esistono diverse convenzioni:

- Se i pacchetti **iniziano con la stringa lib**:
non terminano con la stringa **-dev**
non terminano con la stringa **-dbg**
non terminano con la stringa **-doc**

Sono **librerie dinamiche**.

Eccezione notevole: LibreOffice (pacchetto libreoffice).

- Se i pacchetti **iniziano con la stringa lib e terminano con la stringa -dev**. Sono **librerie statiche**
- Se i pacchetti terminano con **"-doc"** corrispondo ad **documentazione relativa ad uno specifico pacchetto**
- Se i pacchetti terminano con la stringa **"-dbg"** allora contengono le tabelle dei simboli di software specifici

Stampa metadati di un pacchetto

Dopo aver trovato il nome esatto di un pacchetto è possibile stampare i metadati con il comando **"apt show"**

```
apt show PACCHETTO1 PACCHETTO2
```

Dipendenze dirette

Le **dipendenze dirette** di un pacchetto binario P sono i pacchetti binari P1, P2, ..., PN di cui è necessario l'installazione per poter eseguire il software P

Diciamo che senza i pacchetti P1, P2, PN non si potrebbe avviare il software:

```
apt depends PACCHETTO1, PACCHETTO2
```

Ecco la spiegazione del risultato del comando:

```
apt depends bash
```

```
studente@debian:~$ apt depends bash
bash
  Pre-dipende: libc6 (>= 2.15)
  Pre-dipende: libtinfo6 (>= 6)
  Dipende: base-files (>= 2.1.12)
  Dipende: debianutils (>= 2.15)
  Va in conflitto: bash-completion (<< 20060301-0)
  Raccomanda: bash-completion (>= 20060301-0)
  Consiglia: bash-doc
  Sostituisce: bash-completion (<< 20060301-0)
  Sostituisce: bash-doc (<= 2.05-1)
```

- **"Pre-dipende"**: forza l'installazione completa dei pacchetti utili alla installazione di bash, prima della installazione di bash stesso
- **"Dipende:"** forza l'installazione completa dei pacchetti utili alla installazione di bash, durante della installazione di bash stesso
- **"Va in conflitto"**: blocca l'installazione di quel pacchetto, perchè se si installasse andrebbe a distruggere il pacchetto già esistente creando degli effetti disastrosi.

- **"Raccomanda"**: Esegue l'installazione di bash-completion di quella versione indicata
- **"Consiglia"**: Per una **migliore fruizione del software offerto dal pacchetto bash**, il gestore dei pacchetti consiglia bash-doc.
- **"Sostituisce"**: indica al gestore dei pacchetti di non considerare un errore il fatto che bash **sovrascriva /usr/share/doc e /usr/bin**

Contiene anche dei caratteri particolari:

- **"|"** (pipe): Introduce l'OR di più pacchetti, **se almeno uno di questi due pacchetti è installato, le dipendenze sono soddisfatte. Se nessun pacchetto installato, APT installa il primo nell'elenco (il backend Gstreamer).**
- **"<>"** I caratteri <> delimitano il nome di un pacchetto virtuale, cioè non realmente installato nel sistema.

Dipendenze inverse

Le dipendenze **inverse** di un pacchetto binario P sono i pacchetti binari P1, P2,..., PN che richiedono l'installazione di P per poter eseguire il proprio software

Diciamo che scopriamo tutti i software che fanno utilizzo del pacchetto P.

```
apt rdepends PACCHETTO01 PACCHETTO02
```

Grafo delle dipendenze dirette

Per calcolare e disegnare il grafo delle dipendenze dirette di un pacchetto bisogna usare due comandi:

- "**debtree**" (pacchetto **debtree**): calcola il grafo delle dipendenze di un pacchetto
- "**dot**" (pacchetto graphviz): disegna grafi

⚠ E' necessario prima scaricarli (debtree, graphviz) ⚠

Ora vediamo come calcolare il grafo delle dipendenze di bash

Per **calcolare il grafo** delle dipendenze dirette è molto semplice:

```
debtree --no-conflicts --no-recommends bash > bash.dot
```

Il risultato è un **file testuale** (in formato .dot) rappresentante il grafo delle dipendenze (togliamo i conflicts e i recommends per questione di spazio)

Per **disegnare il grafo**:

```
dot -Tsvg bash.dot > bash.svg
```

Il risultato è un'immagine vettoriale (**SVG**) mostrando il grafo delle dipendenze di **bash**

Visualizzazione del grafo

Per visualizzare il grafo appena creato possiamo utilizzare un programma che si chiama "**inkscape**"

⚠ E' necessario prima scaricare il pacchetto (inkscape) ⚠

```
inkscape bash.svg
```

Ricerca del contenuto dei pacchetti

Tramite il comando **apt-file** permette di effettuare ricerche più sofisticate sui pacchetti.

⚠ E' necessario prima scaricare il pacchetto (apt-file) ⚠

Per **poter essere utilizzato**, è **necessario** aggiornare periodicamente l'**indice locale** che **associa un file al nome del pacchetto che lo contiene visto che proviene da un sito Web**.

Per farlo è necessario scrivere:

```
apt-file update
```

Elenco file di un pacchetto

Per capire quali file sono forniti da un pacchetto software potete usare l'argomento **list** con l'opzione **-x**

```
apt-file list -x '^libc6$'
```

⚠ Funziona anche su pacchetti non scaricati

Ricerca del pacchetto a partire dal file

Per capire quale pacchetto fornisce un file specifico potete usare l'argomento **search** con l'opzione **-x** (ricerca di espressione regolare):

```
apt-file search -x '^/usr/bin/top$'
```

Pacchetti binari speciali

Metapacchetto

E' un **pacchetto che installa pochi file** (tipicamente, documentazione) e dipende direttamente da un gruppo di pacchetto. Il suo scopo primario è quello di installare il gruppo di pacchetti.

Per capire se è un metapacchetto basta:

- Fare la **lista di file** che contiene (**devono essere una manciata**) con "apt-file list -x ..."
- Fare la **lista di dipendenze** che contiene (**devono essere una marea**) "apt-file search -x ..."

Pacchetto virtuale

Un **pacchetto virtuale** è un **pacchetto fantasma** che, di per sé, **non fornisce file**, bensì una lista di pacchetti che possono soddisfarlo

In **questo aspetto differisce dal metapacchetto** (che, comunque, installa file sul file system).

Un **pacchetto virtuale** è un **modello generico di pacchetto** che **fornisce una data funzionalità** (editor, JVM, browser Web, ...) ed indica quali pacchetti offrono tale funzionalità.

Gestione delle alternative (domanda chiesta all'esame)

Tramite il comando "**update-alternatives**" permette di elencare tutti i pacchetti virtuali e l'alternativa di default corrispondente, elencare tutte le alternative e cambiare quella di default

- Per visualizzare l'elenco di tutti i pacchetti virtuali è possibile:

```
update-alternatives --get-selections
```

- Per visualizzare l'elenco di tutti i pacchetti alternativi già installati per un dato pacchetto:

```
update-alternatives --list NOMEpacchettoVIRTUALE
```

- Per modificare l'alternativa di default attuale per un determinato pacchetto:

```
update-alternatives --config NOMEpacchettoVIRTUALE
```

e selezionare il pacchetto che si vuole avere.

Per esempio vogliamo cambiare il nostro editor di default, andiamo a vedere cosa abbiamo di default

```
update-alternatives --get-selections | grep "editor"

# risultato
editor /usr/bin/emacs
...
```

Ottimo, vediamo quali abbiamo disponibili già scaricati:

```
update-alternatives --list editor

# risultato
/bin/nano
/usr/bin/emacs
...
```


(in caso non soddisfano quelli presenti nella lista di già scaricati, è possibile andare a cercare altri editor tramite il apt install editor e tu usciranno tutti gli editor che si possono scaricare).

Ora modifichiamo il nostro editor:

```
update-alternatives --config editor
```

e selezioniamo quello che vogliamo