Bash Guide & Command Cheatsheet — A dummy-to-dummy summary

- Micholas Magi, nicholas.magi@studio.unibo.it,
 - con il contributo di Gioele Foschi gioele.foschi@studio.unibo.it
- Corso di Sistemi Operativi @ CdL in Ingegneria e Scienze Informatiche,
 Università di Bologna Campus di Cesena.
- Riassunto e schematizzazione delle dispense del prof. Vittorio Ghini.
- Scritto su: <u>Obsidian</u> consigliato il suo utilizzo poiché alcuni contenuti non sono direttamente fruibili da altri interpreti di Markdown.
- Nozioni per uso del terminale: Metacaratteri
- Nozioni per uso del terminale: Espansioni
- Nozioni per uso del terminale: Variabili
 - Variabili note: PATH
 - Variabili note: \$BASHPID e \$\$
 - Riferimenti indiretti a variabili
 - Manipolare il contenuto della variabile
 - Lunghezza del contenuto di una variabile
 - Rimozione di suffissi
 - Rimozione di prefissi
 - Sostituzione
 - Variazioni di comportamento: sostituzione TOTALE
 - Variazioni di comportamento: sostituzione all'INIZIO
 - Variazioni di comportamento: sostituzione alla FINE
 - Substring
 - Espansione di nomi di variabili corrispondenti ad un prefisso
- Nozioni per uso del terminale: Permessi di file e directory
- Nozioni per uso del terminale: Wildcards
 - Wildcard elenco
- Nozioni per uso del terminale: Parameter Expansion
- Nozioni per uso del terminale: Valutazione Aritmetica
 - Nozioni per uso terminale: Espressioni condizionali
 - Valori di verità

- Operatori logici
- Versioni
- Operatori utili
 - Su FILES
 - Su STRINGHE
- Nozioni per uso del terminale: Liste di comandi
- Nozioni per uso del terminale: Compound Commands
 - Cicli costrutto iterazione
 - If costrutto selezione
- Nozioni per uso del terminale: Command substitution
- Nozioni per uso del terminale: Ridirezionamenti di Stream I/O
- Nozioni per uso del terminale: Raggruppamento di comandi
- Nozioni per uso del terminale: Processi
 - Processi in foreground
 - Processi in background
 - Jobs
 - Job control
 - Processi Zombie & Orfani
- Nozioni per uso del terminale: Precedenza degli operatori
 - Terminatori di una seguenza di comandi
 - "Concatenatori" di una sequenza di comandi
 - Ordine
- Cheatsheet comandi
 - Generics
 - Lettura & gestione file descriptor
 - Visualizzare i file descriptor associati ad un processo
 - read
 - exec
 - Manipolazione di stringhe & informazioni testuali
 - head
 - tail
 - tee
 - cut
 - grep
 - sed Stream Editor
 - Gestione processi
 - disown

- nohup
- find

Nozioni per uso del terminale: Metacaratteri

METACARATTERI	SIGNIFICATO
> >> <	redirezione I/O
\	pipe
* ? []	wildcards
`command`	command substitution (use backticks)
;	esecuzione sequenziale - separatore di comandi
. \ &&	esecuzione condizionale
()	raggruppamento comandi
&	esecuzione in background
11 11 1	quoting
#	commento
\$	espansione di variabile
\	carattere di escape *
<<	"here document"

Nozioni per uso del terminale: Espansioni

In ordine di effettuazione

ESPANSIONE	ESEMPIO
1) history expansion	!123
2) brace expansion	a{damn,czk,bubu}e
3) tilde expansion	~/nomedirectory
4) parameter and variable expansion	\$1 \$? \$! \${var}
5) arithmetic expansion	\$(())
6) command substitution LTR ¹	\$()

ESPANSIONE	ESEMPIO
7) word spitting	
8) pathname expansion	* ? []
9) quote removal	quoting

1 — LTR: Left To Right

Nozioni per uso del terminale: Variabili

ESEMPIO	SIGNIFICATO
MYVAR=PAROLA	MYVAR ha valore PAROLA; unico modo di assegnare un valore ad una variabile, non mettere spazi prima/dopo o in mezzo!
echo \${MYVAR}	stampo il valore di MYVAR, quindi PAROLA.
echo \${MYVAR}x	stampo il valore di MYVAR seguito dal carattere x , quindi visualizzerò PAROLAx .
echo \$MYVAR	stampo il valore di MYVAR, quindi PAROLA.
echo \$MYVARx	la shell non è in grado di trovare una variabile MYVARx , quindi stampa stringa vuota .
echo \$MYVAR	la shell individua correttamente la variabile MYVAR, quindi visualizzerà PAROLA x.

Variabili note: PATH

PATH è una variabile d'ambiente che contiene i percorsi assoluti di alcuni eseguibili utilizzati molto spesso.

(i) Info

Esempio di un possibile valore di path:

/bin:/sbin:/usr/bin:/usr/local/bin:/home/nickolausen

Variabili note: \$BASHPID e \$\$

Ogni processo è identificato da un codice numerico identificativo detto **PID** (**P**rocess **ID**entifier);

Per visualizzare il PID di un processo si fa riferimento alla variabile \$\$;

```
Shell
1 > echo $$
2 2606
```

△ Eccezione di comportamento!

\$\$ si riferisce al PID della shell padre di un processo. Se voglio vedere il PID di una sub-shell lanciata da un gruppo di comandi, devo utilizzare la variabile \$BASHPID!

\$BASHPID è definita solo in bash e solo per le versioni di bash > 4.0!

```
shell

echo "PID fuori $$" ; ( echo "PID dentro $$" ; echo -n "" )

# Output SENZA USO DI $BASHPID

PID fuori 1760

PID dentro 1760
```

```
Shell

1 echo "PID fuori $$" ; ( echo "PID dentro $BASHPID" ; echo -n "" )

2 
3 # Output CON USO DI $BASHPID

4 PID fuori 1760

5 PID dentro 2042
```

Riferimenti indiretti a variabili

Operatore !:

```
se IDX=1, ${!IDX} == $1;
se IDX=2, ${!IDX} == $2;
se IDX=3, ${!IDX} == $3;
```

• ecc...

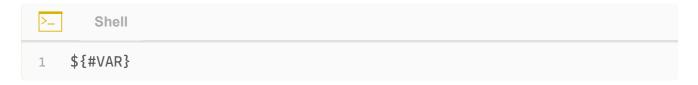
Manipolare il contenuto della variabile

Marning

Le seguenti espansioni *non vanno a modificare la variabile*, ma mostrano solamente la modifica apportata.

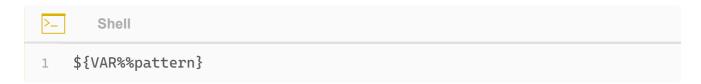
Supponiamo di avere la variabile VAR="[13] qualcosa con [0] fine"

Lunghezza del contenuto di una variabile



- Ottengo la lunghezza della stringa memorizzata in VAR;
- echo \${#VAR} stampa in output: 28

Rimozione di suffissi



- Rimuovo il più lungo suffisso che fa match con la stringa originale
- echo \${VAR%%]*} stampa in output: [13

```
Shell

1 ${VAR%pattern}
```

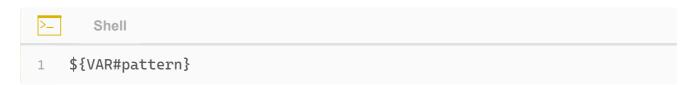
- Rimuovo il *più corto* suffisso che fa match con la stringa originale
- echo \${VAR%]*} stampa in output: [13] qualcosa con [0

Rimozione di prefissi



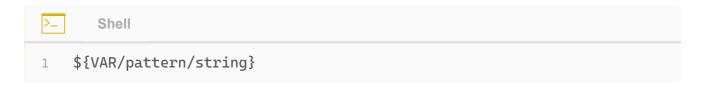
Rimuovo il più lungo prefisso che fa match con la stringa originale

echo \${VAR##*[} stampa in output: 0] fine



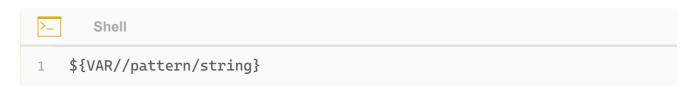
- Rimuovo il più corto prefisso che fa match con la stringa originale
- echo \${VAR#*[} stampa in output: 13] qualcosa con [0] fine

Sostituzione



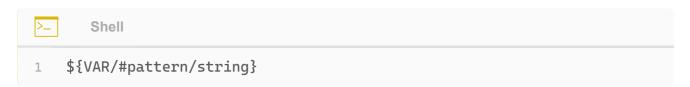
• Cerca nel contenuto di VAR la *sottostringa più lunga* che fa match con il pattern fornito (anche con <u>Wildcards</u>) e lo *sostituisce* con string

Variazioni di comportamento: sostituzione TOTALE



Come per una normale sostituzione, ma questa viene effettuata su **tutte le occorrenze di** pattern, non solo sulla prima trovata.

Variazioni di comportamento: sostituzione all'INIZIO



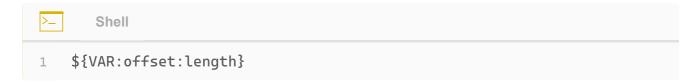
Come per una normale sostituzione, ma questa viene effettuata solo se pattern si trova all'inizio della variabile.

Variazioni di comportamento: sostituzione alla FINE

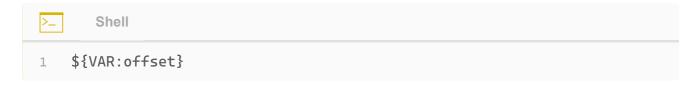


Come per una normale sostituzione, ma questa viene effettuata solo se pattern si trova alla fine della variabile.

Substring



 Mostra la sottostringa lunga length che parte dal carattere numero offset del contenuto di VAR



 Mostra la sottostringa che parte dal carattere numero offset del contenuto di VAR

Espansione di **nomi di variabili** corrispondenti ad un prefisso

```
Shell

1 ${!VarNamePrefix*}
```

Restituisce un elenco con tutti i nomi delle variabili il cui nome inizia con il prefisso specificato VarNamePrefix.

Esempio: data l'esistenza delle seguenti variabili



Per vedere le variabili il cui nome inizia con BASH_AR devo digitare il comando:



```
2
```

Output: BASH_ARGC BASH_ARGV

Nozioni per uso del terminale: *Permessi di file e directory*

Ogni file appartiene ad un utente e ad un gruppo di cui l'utente fa parte: per cambiare owner, usare



I permessi di un file sono identificati dai seguenti codici **letterali** e **ottali**, divisi nelle 3 categorie assegnabili: USER, GROUP e OTHERS.

USER			GROL	JP		OTHE	RS	
R	W	X	R	W	X	R	W	X
4	2	1	4	2	1	4	2	1

Esempio di assegnazione contemporanea di più permessi **mediante formato numerico**:



Dove:

- 7 = 4 (lettura) + 2 (scrittura) + 1 (esecuzione) per livello USER;
- 6 = 4 (lettura) + 2 (scrittura) per livello GROUP;
- 4 (lettura) per OTHERS;

Nozioni per uso del terminale: Wildcards

- * -> sostituito con una qualunque stringa di caratteri;
- ? -> sostituito con un qualunque carattere;

• [elenco] -> sostituito con un qualunque carattere specificato in elenco;

Wildcard elenco

WILDCARD	SOSTITUZIONE
[[:digit:]]	una sola cifra, 0-9
[[:upper:]]	un solo carattere MAIUSCOLO
[[:lower:]]	un solo carattere minuscolo
[c-f]	un solo carattere compreso tra 'c' ed 'f'; in questo caso, {c, d, e, f}
[1-7]	un solo carattere compreso tra 1 e 7; in questo caso, {1, 2, 3, 4, 5, 6, 7}
[abk]	un solo carattere tra 'a', 'b' o 'k'

Nozioni per uso del terminale: Parameter Expansion

Siamo sulla bash, chiamiamo il nostro script passandogli qualche argomento:

```
Shell

1 ./faiqualcosa.sh argo1 mamma2 soreta4
```

Dentro a faiqualcosa.sh possiamo accedere a diverse informazioni identificate come:

- \$#: numero di argomenti ricevuti (*nell'esempio, 3*);
- \$0 : nome del processo in esecuzione (*nell'esempio, ./faiqualcosa.sh *)
- \$1 primo argomento, \$2 secondo argomento... (nell'esempio, \$1 = argo1, \$2 = mamma2, \$3 = soreta4)
- \$*: tutti gli argomenti ricevuti, concatenati e separati da spazi bianchi (nell'esempio, argo1 mamma2 soreta4)
- \$@: come per \$*, ma gli argomenti sono quotati separatamente (nell'esempio, "argo1" "mamma2" "soreta4")

In particolare:

```
# Se quoto le due variabili, ottengo:
# "$*" ---> "$1 $2 $3 ... $n"
# "$0" ---> "$1" "$2" "$3" ... "$n"
```

Nozioni per uso del terminale: *Valutazione Aritmetica*

TUTTA LA RIGA — Assegno alla variabile NUM il risultato di 3+2 :

```
Shell

1 (( NUM=3+2 ))
```


Non usare \$ prima dell'espressione!

PARTE DI RIGA — faccio riferimento, per esempio, ad una variabile PIPP05

```
>_ Shell

1 echo PIPPO$((3+2))
```

Le espressioni aritmetiche possono contenere:

- operatori aritmetici (+, -, *, /, %)
- Z assegnamenti
- parentesi tonde per modificare la precedenza dei calcoli

(i) Info

Le espressioni aritmetiche possono essere usate anche come condizione di while e if.

Nozioni per uso terminale: *Espressioni condizionali*Valori di verità

- ullet true : "exit status di un command " =0
- false: "exit status di un command " eq 0

Operatori logici

NOT: !

AND: && , -a

OR: ||, -o

Versioni

Marning

In nessuna versione è supportato:

- X WORD SPLITTING
- X BRACE EXPANSION
- X PATHNAME EXPANSION
- X INSERIMENTO COMANDI (ad eccezione della command subsitution con "per generare operandi, non operatori!)
- 1. Con doppie parentesi quadre [[condiz]] enhanced brackets
- SI a tutte le versioni di operatori logici
- SI a composizione di espressioni mediante parentesi
- SI ad andata a capo con carattere backslash (\)
- 2. Con singole parentesi quadre [condiz] single brackets o mediante istruzione test: test condiz
- X NO versioni degli operatori logici C-like && e | |
- X NO composizione di espressioni mediante parentesi
- X NO ad andata a capo con carattere backslash (\)

Operatori utili

Su FILES

OPZIONE	SPIEGAZIONE	
-d file	True if file exists and is a directory.	

OPZIONE	SPIEGAZIONE
-e file	True if file exists.
-f file	True if file exists and is a regular file.
-h file	True if file exists and is a symbolic link.
-r file	True if file exists and is readable.
-s file	True if file exists and has a size greater than zero.
-t fd	True if file descriptor fd is open and refers to a terminal.
-w file	True if file exists and is writable.
-x file	True if file exists and is executable.
-O file	True if file exists and is owned by the effective user id.
-G file	True if file exists and is owned by the effective group id.
file1 -nt file2	True if file1 is newer (according to modification date) than file2, or if file1 exists and file2 does not.
file1 -ot	True if file1 is older (according to modification date) than
file2	file2, or if file2 exists and file1 does not.

Su STRINGHE

OPZIONE	SPIEGAZIONE
-z string	True if the length of string is zero.
-n string	True if the length of string is non-zero.
<pre>string1 == string2, string1 = string2, string1 -eq string2</pre>	True if the strings are equal.
string1 != string2, string1 -ne string2	True if the strings are not equal.
<pre>string1 < string2, string1 -lt string2</pre>	True if string1 sorts before string2 lexicographically.
<pre>string1 > string2, string1 -gt string2</pre>	True if string1 sorts after string2 lexicographically.

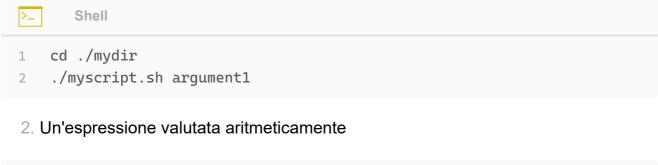
(i) Info

Esistono anche le varianti -le (*less or equal to*) e -ge (*greater or equal to*).

Nozioni per uso del terminale: Liste di comandi

In una command line posso scrivere:

1. Un semplice comando



```
Shell

1 (( VAR=(5+4)*(${VAR}-1)))
```

3. Una sequenza di comandi connessi da | (pipe)

```
Shell

cat $FILE | grep thisword
```

- 4. Una sequenza di comandi condizionali, in particolare
- **4.1.** Eseguo un comando B **se e solo se** il comando A è andato a buon fine (exit status == 0);

[per esempio: lancio l'eseguibile myscript.exe SE E SOLO SE la sua compilazione è andata a buon fine]

```
Shell

1 gcc myscript.c && ./myscript.exe
```

4.2 Eseguo un comando B **se e solo se** il comando A è terminato con un errore (exit status != 0);

[per esempio: creo la cartella mysecretfiles SE E SOLO SE provando ad entrarci non ha avuto successo]

Shell

1 cd ./mysecretfiles || mkdir mysecretfiles

5. Un raggruppamento di comandi

Shell

1 (cat file1.txt; cat file2.txt)

6. Un'espressione condizionale

>_ Shell

1 [[-e file.txt && \$1 -gt 13]]

(i) Info

L'exit status di una lista di comandi corrisponde all'exit status dell'ultimo comando eseguito!

Nozioni per uso del terminale: *Compound Commands*

Cicli — costrutto iterazione

1. Equivalente di un foreach

Shell

1 for FILE in `ls ./Documents` ; do echo \$FILE ; done

2. Equivalente di un for

Shell

for ((IDX=1;IDX<=\$#;IDX=IDX+1)); do echo \${!IDX}; done</pre>

[nell'esempio: stampo a video tutti gli argomenti ricevuti dallo script in esecuzione]

3. Costrutto while

```
Shell

while read ROW; do
    echo "Ho letto $ROW"

done
```

N.B.: Posso comporre anche in diverso modo le condizioni di un ciclo, ad esempio:

```
IDX=1
while read ROW; [[ $? == 0 && $IDX -leq 10 ]]; do
echo "Idx:${IDX}\t${ROW}"
((IDX=$IDX+1))
done
```

[nell'esempio: leggo al massimo 10 righe da stdin]

If — costrutto selezione

Nozioni per uso del terminale: *Command* substitution

Utilizzo dei **backticks** *(o backquotes)* (``): cattura l'output di un programma - utilizzato per memorizzare i risultati di uno script in una variabile.

```
Shell

1 OUT=`.\printnumber.sh`
2 echo "Catturato l'output $OUT"
```

Utilizzo di double quotes (""):

- X NO; come separatore tra comandi
- X NO sostituzione wildcards

- SI visualizzazione contenuto variabili
- SI esecuzione comandi (se racchiusi da ``)

```
Shell

1 echo "Dear $USER today is `date`"

2 
3 stdout: Dear nickolausen today is Fri Dec 13 11:20:55 CET 2024
```

Utilizzo di single quotes ("):

- X NO; come separatore tra comandi
- X NO sostituzione wildcards
- X NO visualizzazione contenuto variabili
- X NO esecuzione comandi (anche se racchiusi da ``)

```
Shell

1 echo 'Dear $USER today is `date`'
2 stdout: Dear $USER today is `date`
```

Nozioni per uso del terminale: *Ridirezionamenti di Stream I/O*

RIDIREZIONAMENTO	SIGNIFICATO		
<	riceve input da file.		
>	manda stdout verso un file, eliminando il vecchio contenuto del file.		
>>	manda stdout verso un file, aggiungendo in coda al vecchio contenuto del file.		
	manda stdout del primo comando come stdin del secondo.		

Per terminare input da tastiera: Ctrl + D

• Ridirezionamento di input e output possono essere fatti contemporaneamente:

```
Shell

1 ./myscript.sh < ./input.txt > ./output.txt

2 
3 # oppure, analogamente:
4 ./myscript.sh > ./output.txt < ./input.txt</pre>
```

• Ridirezionamento di stdout e stderr possono essere fatti contemporaneamente:

```
Shell

1 ./myscript.sh &> ./out.txt
```

 Ridirezionamento di stdout e stderr possono essere fatti in un colpo solo su due file diversi:

```
Shell

1 ./myscript.sh 2> ./error.txt > ./output.txt
```

• È possibile ridirigere uno stream N sullo stream M tramite il comando:

```
>_ Shell

1 N>&M
```

• È possibile ridirigere contemporaneamente stderr e stdout di un programma program1 sullo stdin di un programma program2 tramite pipe:

```
Shell

1 ./program1.sh |& ./program2.sh
```

Nozioni per uso del terminale: Raggruppamento di comandi

Una sequenza di comandi racchiusa da una coppia di parentesi tonde viene eseguita in una sub-shell. L'exit status di quella sequenza corrisponde all'exit status dell'ultimo comando eseguito.

In caso contrario, non viene creata nessuna subshell.

Esempio di sequenza di comandi:

```
Shell

1 (pwd; ls -al; whoami) > ./out.txt
```

Marning

Durante l'esecuzione, stdin, stdout e stderr dei singoli comandi vengono concatenati.

Nozioni per uso del terminale: Processi

Processi in foreground

- Processi che controllano il terminale da cui sono stati lanciati: ne condividono stdin, stdout e stderr; non permettono l'esecuzione di altri programmi.
- In ogni istante di tempo al più 1 processo può essere in foreground.

Processi in background

Processi eseguiti in parallelo rispetto all'esecuzione della bash; detengono una
copia dei file descriptor associati alla shell che li ha lanciati, quindi condividono
stdin / stderr / stdout; questo comporta che la chiusura del terminale causi a
sua volta la terminazione di tutti i processi — affinché questi possano continuare a
prescindere dalla vita del terminale, occorre "sganciarli" da esso.

Jobs

Processi in background o sospesi solo figli di quella shell.

Job control

Spostamento di un processo background ↔ foreground

Processi Zombie & Orfani

Si parla di queste 2 categorie di processi quando viene chiamato il comando wait (vedi sotto, in Command Cheatsheet).



Ricordiamo che ogni processo è composto da un PID (identificativo del processo) + un PCB (*Process Control Block*, insieme di attributi del processo)

- 1. Se la shell padre termina senza aver effettuato una chiamata wait \$PROC_PID, allora il processo figlio viene definito **processo orfano**. Tutti i processi orfani vengono adottati dal processo **init**, colui che detiene il PID = 1, che provvederà a chiamare wait per i nuovi arrivati così da far rilasciare il PCB.
- 2. Se la shell figlia termina prima dell'esecuzione della shell padre, allora il figlio viene detto **processo zombie**. Il processo è terminato, ma il suo PCB è ancora presente nelle tabelle del sistema operativo. Questo viene eliminato solo quando viene effettuata una chiamata dal padre a wait \$PROC_PID.

Nozioni per uso del terminale: *Precedenza degli* operatori

Terminatori di una sequenza di comandi

```
Shell

1 ; & newline # andata a capo
```

"Concatenatori" di una sequenza di comandi



Ordine



Ordine per precedenza decrescente

```
    { ...; }, ( ...) — con le parentesi graffe è obbligatorio il ; finale!
    |
    &&, |
    & &, ;
```

Cheatsheet comandi

Generics

ESEMPIO	SIGNIFICATO
source ./myscript.sh	Esegue myscript.sh senza creare una subshell dedicata. Le variabili create dentro myscript.sh sono visibili anche dalla shell chiamante. Equivale a/myscript.sh .
unset MYVAR	Elimino MYVAR, che sia vuota oppure no.
history	Mostra sul terminale lo storico dei comandi eseguiti, associando a ciascuno un numero immutabile che lo identifica .
!181	Lancio il comando identificato dal numero 181 (guarda history).
set	Lanciato senza parametri, mostra tutte le variabili (locali e d'ambiente) della shell corrente e tutte le funzioni implementate.
set -a	Specifica che tutte le variabili create da quel momento in poi verranno rese d'ambiente, ereditate quindi dalle shell figlie. Comportamento contrario: set +a . N.B. : Se voglio dichiarare una variabile locale dopo aver eseguito set -a , allora utilizzo il comando export -n MYVAR .
set +o history	Disabilita la memorizzazione nella history di ulteriori comandi eseguiti. Comportamento contrario: set -o history.
pwd	"Print Working Directory": visualizza il percorso corrente.
cd mydir	"Change directory", abbreviativo di chdir, naviga nella cartella mydir.
mkdir nuovadir	"Make directory", crea una nuova cartella nuovadir.
rmdir todelete	"Remove directory", elimina todelete SOLO SE QUESTA È VUOTA. Equivalentemente (e senza condizioni) si può optare per rm -rf todelete ("remove -recursive -forced todelete")

ESEMPIO	SIGNIFICATO
echo MSG	Stampa in stdout MSG. Se specificato -n , non mette il carattere \n a fine messaggio (utile per la stampa nei file).
cat myfile.txt	Stampa in stdout il contenuto del file myfile.txt.
env	Visualizza in stdout tutte le variabili d'ambiente.
which python	Visualizza il percorso dell'eseguibile python.
mv src dst	"Move", sposta il file src in dst. Spesso utilizzato per rinominare files/directories.
ps aux	Stampa informazioni sui processi in esecuzione.
du mydir	"Disk Usage": stampa l'utilizzo del disco della cartella mydir.
kill -9 PID	Termina il processo con ID PID .
killall nomeProc	Elimina tutti i processi di nome nomeProc .
bg	"Background": manda il job corrente in background.
fg	"Foreground": ripristina il job più recente in primo piano.
df	"Disk free": mostra spazio libero dei filesystem montati.
touch newfile.txt	Crea un file newfile.txt.
more myfile.txt	Mostra poco alla volta il contenuto di myfile.txt
man grep	Mostra la documentazione del comando grep (preso come esempio).
true	Restituisce exit status 0 (vero).
false	Restituisce exit status 1 (falso).

Lettura & gestione file descriptor

Visualizzare i file descriptor associati ad un processo

Ricordiamo che in /proc/ esiste una subdirectory per ogni processo in esecuzione; posso riferirmi alla subdirectory del processo corrente con /proc/\$\$ (\$\$ = PID del processo corrente).

✓ I file descriptor associati ad un processo sono collocati nella cartella /proc/\$\$/fd/

read



Shell

1 read RIGA

Di default, legge il contenuto in stdin e lo memorizza in RIGA ; se vengono specificate più variabili, il contenuto letto viene spezzato in più parole - ciascuna "incastrata" nella variabile corrispondente ($prima\ parola \rightarrow prima\ variabile,\ seconda\ parola \rightarrow seconda\ variabile\ ecc...$). Nel caso in cui vengono dichiarate meno variabili rispetto alle parole da leggere, **l'ultima variabile contiene il testo mancante**

OPZIONI UTILI	
-u \$FD	specifica il file descriptor FD da cui leggere.
-N 4	legge esattamente 4 caratteri da stdin.
-r	il carattere backslash (\) non viene interpretato come interruttore di linea.

exec

Ridireziona un file descriptor su un altro.



Nell'esempio, ridireziona qualsiasi messaggio stampato su stdout all'interno di file fino al comando exit . Può essere utilizzato in diverse modalità:

MODO APERTURA	Utente sceglie il numero associabile al file descriptor (n: numero identificativo del file descriptor)	Sistema sceglie file descriptor libero
Read-only	exec n< ./myfile.txt	<pre>exec {MYVAR}< ./myfile.txt</pre>
Write-only	exec n> ./myfile.txt	<pre>exec {MYVAR}> ./myfile.txt</pre>
Append	exec n>> ./myfile.txt	<pre>exec {MYVAR}>> ./myfile.txt</pre>

MODO APERTURA	Utente sceglie il numero associabile al file descriptor (n : numero identificativo del file descriptor)	Sistema sceglie file descriptor libero
Read&Write	exec n<> ./myfile.txt	<pre>exec {MYVAR}<> ./myfile.txt</pre>

(i) Ricorda!

Bash assegna ad alcuni numeri dei file descriptor di default:

- ullet 0 o stdin
- 1 \rightarrow stdout
- 2 → stderr

Marning

Qualunque sia la modalità di apertura di un file descriptor, questo deve essere chiuso con il comando:

Bash

```
1 exec n>&-
2 # n = identificativo del file descriptor in questione
3
4 # oppure, se aperto tramite variabile
5 exec {MYVAR}>&-
```

Manipolazione di stringhe & informazioni testuali

head

Seleziona un certo numero di righe di un testo a partire dal suo inizio.

ARGS UTILI	ESEMPIO	DESCRIZIONE
-n \$NUM	cat ./myfile.txt \ head -n 2	Indica il numero \$NUM di righe da selezionare.
\$FILE	head -n 4 \$FILE	Indica il file \$FILE da cui selezionare le prime 4 righe.

ARGS UTILI	ESEMPIO	DESCRIZIONE
-c \$NUM	cat ./myfile.txt \ head -c 10	Stampa i primi \$NUM caratteri del testo da manipolare.

tail

Seleziona un certo numero di righe di un testo a partire dalla sua fine.

ARGS UTILI	ESEMPIO	DESCRIZIONE
-n \$NUM	<pre>cat ./myfile.txt \ tail -n 2</pre>	Indica il numero \$NUM di righe da selezionare.
\$FILE	tail -n 4 \$FILE	Indica il file \$FILE da cui selezionare le ultime 4 righe.
-c \$NUM	cat ./myfile.txt \ tail -c 10	Stampa gli ultimi \$NUM caratteri del testo da manipolare.
-r	cat ./myfile.txt \ tail -r -n 5	"Reversed", inverte l'ordine dell'output.

tee

Inoltra il contenuto di stdin su più file contemporaneamente.

```
Shell

cat ./myfile.txt | tee out1.txt out2.txt
```

Nel seguente esempio, inoltro il messaggio Hello, World! sia a stdout che al file greetings.txt



cut

Seleziona solo una **porzione** del file da manipolare.

ARGS UTILI	ESEMPIO	DESCRIZIONE
-b \ -c from- to,from- to,	cat ./myfile.txt \ cut -c 2- 10,15-20	Indica il range di caratteri (o, equivalentemente, <i>bytes</i>) da selezionare dal testo ricevuto (<i>nell'esempio</i> , <i>seleziono i caratteri compresi tra il secondo e il decimo e tra il quindicesimo e il ventesimo</i>)

Esempio estratto dall'output del comando man cut :

Extract users' login names and shells from the system passwd(5) file as "name:shell" pairs:



grep

Seleziona solo le righe che rispettano il criterio specificato.

```
Shell

cat ./dizionario.txt | grep APPLE
```

Nell'esempio: seleziono la riga dal file dizionario.txt che contiene la parola APPLE

ARGS UTILI	ESEMPIO	DESCRIZIONE
-f \$FILE	grep APPLE -f \$DIZIONARIO	Indica il file da cui leggere le righe. (Se non funziona, mettere il nome del file subito dopo l'espressione da cercare nel file.)
-v	cat ./dizionario.txt \ grep APPLE -v	Inverte la selezione: seleziona tutte le righe che NON contengono la parola <i>APPLE</i> .
-с	cat ./dizionario.txt \ grep APPLE -c	Mostra il numero di righe che rispettano il criterio specificato.
-i	cat ./dizionario.txt \ grep ApPlE -i	Effettua una ricerca case-insensitive.
-m \$NUM	cat ./voti.txt \ grep 18 -m 5	Mostra solo le prime \$NUM righe che soddisfano il criterio specificato.
-n	cat ./voti.txt \ grep 18 -n	Precede ogni riga selezionata con il corrispondente numero di riga nel file.

sed - Stream Editor

Modifica un testo sulla base di una regular expression specificata.

Sintassi tipo:



Dove:

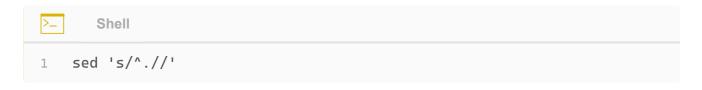
- s indica l'operazione di sostituzione di porzioni di testo;
- DA_TOGLIERE corrisponde alla porzione di testo da rimuovere;
- DA_METTERE corrisponde alla porzione di testo da inserire al posto di DA_TOGLIERE;
- g opzionale: indica che la sostituzione va effettuata su tutto il testo su cui sed sta lavorando ("global") (altrimenti fatta solo sulla prima occorrenza di DA_TOGLIERE in ciascuna riga);
 - numero *opzionale*: indica per quante occorrenze di DA_TOGLIERE deve essere fatta la sostituzione.

Alcuni utilizzi di sed:

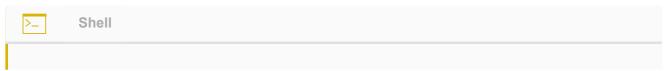
1. Sostituisce **la prima occorrenza** di togli con metti in ciascuna riga del file nomefile.



2. Rimuovere il carattere in prima posizione di ogni linea che si riceve da stdin.



- ^ : inizio linea
- : carattere qualunque
- 3. Rimuovere l'ultimo carattere di ogni linea ricevuta dallo stdin.



- 1 sed 's/.\$//'
- . : carattere qualunque
- \$: fine linea
- 4. Eseguo **due rimozioni insieme** (;) rimuovere il primo e l'ultimo carattere in ogni linea.



1 sed 's/^.//;s/.\$//'

5. Rimuove i primi 3 caratteri ad inizio linea



1 sed 's/...//

6. Rimuove i primi 4 caratteri ad inizio linea.

>_ Shell

1 sed $-r 's/.{4}//'$

7. Rimuove gli ultimi 3 caratteri di ogni linea.

>_ Shell

1 sed -r 's/.{3}\$//'

8. Rimuove tutto eccetto i primi 3 caratteri di ogni linea.

>_ Shell

1 sed -r 's/(.{3}).*/\1/'

9. Rimuove tutto eccetto gli ultimi 3 caratteri di ogni linea.

>_ Shell

1 sed $-r 's/.*(.{3})/\1/'$

10. Rimuove tutti i caratteri alfanumerici in una linea.



ARGS UTILI	DESCRIZIONE	
-r \ -E	Interpreta la regular expression come regular expression moderna (o estesa).	

Gestione processi

ESEMPIO	SIGNIFICATO	
./myscript &	Esegue myscript.sh in background. All'interno della variabile \$! è memorizzato il suo PID. Il carattere & rappresenta un separatore tra comandi — è necessario quindi omettere il ";"	
^Z (<i>Ctrl-Z</i>)	Sospende un processo in foreground.	
^C (Ctrl-C)	Termina un processo in foreground.	
bg	Riprende l'esecuzione in background di un processo sospeso.	
fg %n	Porta in foreground un processo sospeso — %n : indice del job di riferimento. (vedi jobs)	
kill 6152	Elimina il processo con il PID 6152 .	
kill %2	Elimina il processo con l' indice del job = 2 (<i>vedi jobs</i>).	
jobs	Produce una lista numerata dei processi in background o sospesi nella shell corrente — il numero tra parentesi quadre che indica ciascun processo non è il PID ma un indice del job che si usa per gestire il job, premettendo il carattere % . N.B.: Questo comando mostra solo i processi in esecuzione sospesi a partire dalla shell corrente.	
ps -ax	"Process Status": restituisce una visione statica dello stato di tutti i processi di tutti gli utenti ($-a$) che non necessariamente hanno controllato il terminale ($-x$).	
top	Restituisce una visione dinamica dello stato di tutti i processi in esecuzione, aggiornata periodicamente ad intervalli di tempo regolari.	

ESEMPIO	SIGNIFICATO	
wait \$JOBs	Attende la terminazione del comando con Job ID / PID corrispondente, restituendone il risultato. Si possono specificare un elenco di Job o di PID di processi che si vogliono aspettare. Il comando wait termina la sua esecuzione solo quando tutti i processi specificati terminano la loro. Può essere chiamato solo dalla shell che ha lanciato l'esecuzione dei comandi specificati, altrimenti termina subito restituendo come risultato 127. Se non riceve parametri, attende che tutti i processi figli del processo da cui viene chiamato terminino la loro esecuzione.	

⚠ Ricorda!

La variabile \$! memorizza sempre il PID dell'ultimo processo lanciato in background. Fino allo spostamento di un nuovo processo in background, il suo valore rimane **immutato**.

disown

ARGS UTILI	ESEMPIO	DESCRIZIONE
-r	disown -r	Sgancia dalla shell tutti i job running .
-a	disown -a	Sgancia dalla shell tutti i job running e sospesi .
-	disown	Sgancia dalla shell l'ultimo job messo in background.
%jobid	disown %jobid	Sgancia dalla shell il job specificato .

nohup

Esempio:

Shell

1 nohup ./myscript arg1 arg2 &

Lancia uno script in background e lo sgancia dalla shell di partenza. Equivalente di:



- 1 ./myscript.sh arg1 arg2 &
- 2 disown

find

find [path] [options] [expression]

• Serve per cercare dei files in una cartella specificata da [path]

ARGS UTILI	ESEMPIO	DESCRIZIONE
-name	find / -name '*.h'	Cerca <i>tutti</i> i file e directory con quel <i>nome</i> specifico (si possono usare <i>Wildcards</i>)
-maxdepth [n]	find / - maxdepth 2	Cerca <i>tutti</i> i file e directory con una profondità di ricerca di al massimo n
<pre>-mindepth [n]</pre>	find / - mindepth 2	Cerca <i>tutti</i> i file e directory con una <i>profondità di ricerca</i> di al minimo n
-type [f/d]	find / -type f	Cerca solo file f o solo directory d
-exec	find / -exec head -n 1 '{}' \;	Esegue per ciascuno dei file che trova i comandi che seguono. '{}' contiene il nome del file trovato

>_

Shell

- find /usr/ -maxdepth 3 -type f -name '*.h' -exec head -n 1 '{}' \;
- # Search for every .h file inside /usr and at max 3 sub directory and print the first line for each file found