

# OPERAZIONI SU FILE

Modo Apertura	Utente sceglie fd (n è il numero scelto dall'utente)	Sistema sceglie fd libero e lo inserisce in variabile
Solo Lettura	exec n< PercorsoFile	exec \${NomeVar}< PercorsoFile
Scrittura	exec n> PercorsoFile	exec \${NomeVar}> PercorsoFile
Aggiunta in coda	exec n>> PercorsoFile	exec \${NomeVar}>> PercorsoFile
Lettura e Scrittura	exec n<> PercorsoFile	exec \${NomeVar}<> PercorsoFile

NB: i simboli < > >> <> devono essere attaccati (senza spazi) al numero o alla } 141

# esempio:  
# effettuo le lettura dal file mioinput.txt aprendolo in lettura

```
exec $(FD)< /home/vittorio/mioinput.txt
while read -u $FD StringaLetta;
do
echo "ho letto: ${StringaLetta}"
done
```

# esempio:  
# scrivo l'output dei comandi echo sul file miooutput.txt aprendolo in scrittura

```
exec $(FD)> /home/vittorio/miooutput.txt
for name in pippi pippa pippi; do
echo "Inserisco $name"
done
```

echo  
mondo  
allo STDO

→ nella posizione  
1 ora combinare  
e scriviamo  
su FD  
con questo  
sintesi diciamo  
che invece  
di mandare  
sol 1 (= STDO)  
rendiamo

→ per accedere  
ai file

```
studente@lubuntu22:~/BUTTAMI$ exec $(FD)< leggini.txt
studente@lubuntu22:~/BUTTAMI$ read -u $FD VAR
studente@lubuntu22:~/BUTTAMI$ echo $VAR
ho scrittoppolo
studente@lubuntu22:~/BUTTAMI$ read -u $FD VAR
studente@lubuntu22:~/BUTTAMI$ echo $VAR
null
studente@lubuntu22:~/BUTTAMI$ read -u $FD VAR
studente@lubuntu22:~/BUTTAMI$ echo $VAR
ho scrittopiz
studente@lubuntu22:~/BUTTAMI$ read -u $FD VAR
studente@lubuntu22:~/BUTTAMI$ echo $VAR
studente@lubuntu22:~/BUTTAMI$ read -u $FD VAR
studente@lubuntu22:~/BUTTAMI$ echo $VAR
studente@lubuntu22:~/BUTTAMI$
```

→ ora bisogna  
chiudere il  
file descriptor  
exec n>&-

&

exec \${FD}>&-

echo \$! contiene il processo identifier della shell in esecuzione.

read -n 3 nomi file

→ da un numero massimo di  
caratteri da leggere  
con finisce

read -N

lo puoi aggredire leggendo  
tutti i caratteri prestabiliti,  
anche spazi bianchi

## ESPRESSIONI CONDIZIONALI DEI FILE

Alcuni operatori per verificare condizioni su file :

-d file	True if file exists and is a directory.
-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.
-L file	True if file exists and is a symbolic link. (deprecated, see -h)

if [[ .. || .. \$0 .. ]]

is

+ no

Questo comando ti permette di leggere carattere per carattere

## rsi con i comandi di shell (9) Soluzioni

```
23) leggerecaratteri.sh
#!/bin/bash
exec $(FD)< /usr/include/stdio.h
if (( $? == 0 )); then
    NUM=0
    while read -u $FD -N 1 & A ; do
        ((NUM=$NUM+1))
    done
    exec $(FD)>&
    echo ${NUM}
fi
# -N 1 serve per leggere 1 carattere per volta
# -r serve per NON interpretare i incontrati nel file
# come inizio di una sequenza di escape
[[ $? == 0 || ${RIGA} != "" ]] ; do
```

bash:

./nomescript.sh < stdio.txt

"

> " .txt

(riscrivere il file)

>> "

(aggiunge al file)

unset per eliminare contenuto variabile.

unset nome\_vrs

\$(VAR:OFFSET:LENGTH)

string<sup>LENGTH</sup>

OFFSET

program < nome\_file\_input > nome\_file\_output  
o analogamente  
program > nome\_file\_output < nome\_file\_input

per leggere da un file > scrivere  
sopra

Inoltre, in bash si può ridirezionare assieme standard output e standard error su uno stesso file, sovrascrivendo il vecchio contenuto:

program &> nome\_file\_error\_and\_output

→ per avere anche std err

2>

riduzionismo  
solo STDRR

COICE  
RAPPRESENTATIVO

PROGRAM 1

| PROGRAM 2



STDOUT →



STDIN

lanciano contemporaneamente

due processi

Esempio:

while read A B C ; do echo \$B ; done <<FINE

uno due tre quattro

alfa beta gamma

gatto cane

FINE

echo ciao

produce in output

due

beta

cane

ciao

come se leggesse il file  
che c'è sotto <<FINE

testo

FINE

IFS = "#", "\n"

~

VARIABILE

CHE CONTIENE I CARATTERI CHE FUNZIONANO  
DA SEPARATORI DELLE PAROLE NEGLI ELENCHI

TAIL | HEAD

-n ③

nome file da leggere.txt

numero  
di righe  
che verrà visualizzata

tee : per duplicare output

sed :

↳ system editor

```
top/BASH/ESAMI
[BASH/ESAMI]$ sed 's/AL/CUF/g'
```

sostituisce AL con CUF

cut -b i-f  
 \n 11210 fine del cut

```

random.sh
1 #!/bin/bash
2 cont=0
3 bool=0
4 while [[ $bool -eq 0 ]]; do
5   var_support=$((RANDOM))
6   if (( $var_support % 10 == 2 )); then #nel % posso fare un == e semplicemente =
7     echo $RANDOM
8     bool=1
9   else
10    ((cont=$cont+1))
11  fi
12 done
13 echo $cont
14
  
```

$\$* = \$@ \rightarrow \$1 \$2 \$3 \dots \$n$

I parametri  $\$^*$  e  $\$@$  sono uguali quando non quotati dai ", cioè sono la concatenazione separata da blank dei singoli argomenti.

$\$* == \$@ \rightarrow \$1 \$2 \$3 \dots \$n$

I parametri  $\$^*$  e  $\$@$  si comportano diversamente quando sono quotati con " in particolare:

$"\$^*" \rightarrow "\$1 \$2 \$3 \dots \$n"$  quotati tutti gli argomenti assieme

$"\$@"$   $\rightarrow "\$1" " \$2" " \$3" \dots " \$n"$  quotato singolarmente ogni argomento

Comandi della bash (in ordine di importanza)	
pwd	mostra directory di lavoro corrente .
cd <u>percorso_directory</u>	cambia la directory di lavoro corrente .
mkdir <u>percorso_directory</u>	crea una nuova directory nel percorso specificato
rmdir <u>percorso_directory</u>	elimina la directory specificata, se è vuota
ls -alh <u>percorso</u>	stampa informazioni su tutti i files contenuti nel percorso
rm <u>percorso_file</u>	elimina il file specificato
echo <u>sequenza di caratteri</u>	visualizza in output la sequenza di caratteri specificata
cat <u>percorso_file</u>	visualizza in output il contenuto del file specificato
env	visualizza le variabili ed il loro valore
which <u>nomefileesegibile</u>	visualizza il percorso in cui si trova (solo se nella PATH) l'eseguibile
mv <u>percorso_file</u> <u>percorso_nuovo</u>	sposta il file specificato in una nuova posizione
ps aux	stampa informazioni sui processi in esecuzione
du <u>percorso_directory</u>	visualizza l'occupazione del disco.
kill -9 <u>pid</u> processo	elimina processo avente identificativo pid_processo
killall <u>nome_processo</u>	elimina tutti i processi con quel nome nome_processo
bg	ripristina un job fermato e messo in sottofondo
fg	porta il job più recente in primo piano
df	mostra spazio libero dei filesystem montati
touch <u>percorso_file</u>	crea il file specificato se non esiste, oppure ne aggiorna data.
more <u>percorso_file</u>	mostra il file specificato un poco alla volta
head <u>percorso_file</u>	mostra le prime 10 linee del file specificato
tail <u>percorso_file</u>	mostra le ultime 10 linee del file specificato
man <u>nomecomando</u>	è il manuale, fornisce informazioni sul comando specificato
find	cerca dei files
grep	cerca tra le righe di file quelle che contengono alcune parole
read <u>nomivariable</u>	legge input da standard input e lo inserisce nella variabile specificata
wc	conta il numero di parole o di caratteri di un file
true	restituisce exit status 0 (vero)
false	restituisce exit status 1 (non vero)

80

## → LISTA COMANDI

### Parametri a riga di comando passati al programma (4)

#### Parameter Expansion

Come utilizzare in uno script gli argomenti a riga di comando passati allo script

Esistono variabili d'ambiente che contengono gli argomenti passati allo script

- \$# il numero di argomenti passati allo script
- \$0 il nome del processo in esecuzione
- \$1 primo argomento, \$2 secondo argomento, ....
- \$\* tutti gli argomenti passati a riga di comando concatenati e separati da spazi
- \$@ come \$\* ma se quotato gli argomenti vengono quotati separatamente

Esempio: All'interno di uno script posso usarle così :

```
file esempio_script.sh
echo "ho passato $# argomenti alla shell"
echo "tutti gli argomenti sono $*"
```

## → PARAMETRI

### A RIGA DI COMANDO

tento - eseguire (letters.sh)

echo -n letters

    
↓

ti incarna in una riga dello STDOUT

#### Nozioni per uso del terminale: variabili (10)

##### Variabili Vuote vs. Variabili non esistenti

- C'è differenza tra una variabile che esiste ma è vuota ed una variabile che invece non esiste.
- Se una variabile non è mai stata dichiarata allora non esiste.
  - Se a una variabile è stato assegnato come valore la stringa vuota "" allora esiste ma è vuota.

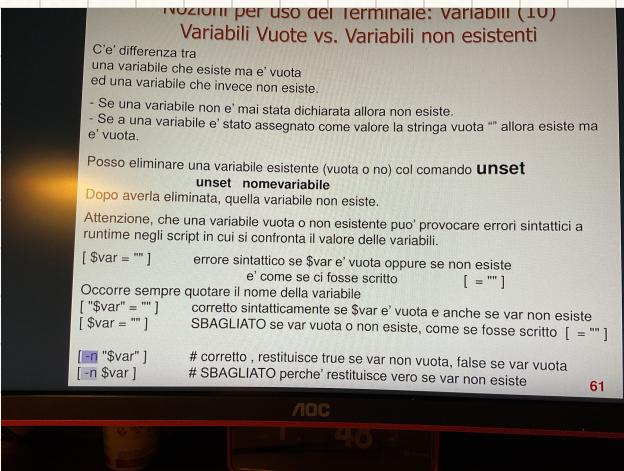
Posso eliminare una variabile esistente (vuota o no) col comando **unset**  
**unset nomevariabile**

Dopo averla eliminata, quella variabile non esiste.

Attenzione, che una variabile vuota o non esistente può provocare errori sintattici a runtime negli script in cui si confronta il valore delle variabili.

```
[ $var = "" ]      errore sintattico se $var è vuota oppure se non esiste
                  e' come se ci fosse scritto           [ = "" ]
Occorre sempre quotare il nome della variabile
[ "$var" = "" ]    corretto sintatticamente se $var è vuota e anche se var non esiste
[ $var = "" ]      SBAGLIATO se var vuota o non esiste, come se fosse scritto [ = "" ]
[ -t "$var" ]      # corretto , restituisce true se var non vuota, false se var vuota
[ -n $var ]        # SBAGLIATO perché restituisce vero se var non esiste
```

-n non esiste



61