

Introduzione alla Programmazione in Bash

E.Mumolo, DIA
`mumolo@units.it`

Bash script

- Formato di uno script in bash:

```
#!/bin/bash
# primo esempio di script
echo $RANDOM
```

Per eseguire lo script come
\$./script

commento

Istruzione di scrittura

Per recuperare il contenuto di una variabile

Variabile d'ambiente
- Lo script deve essere leggibile ed eseguibile per poter essere eseguito
- Oppure si può chiamare la shell direttamente (basta il permesso di lettura):
\$bash script
- Metacaratteri (# * ? > < ! % % | & ; , Apostrofi, Spazio): caratteri con significato particolare
- Gli APOSTROFI hanno un significato particolare:
 - double quote (doppio apice): “ ”
 - single quote (apice singolo) : ‘ ’
 - back quote (apice inverso): ` (si ottiene con AltGr ')

Quoting

- Quoting = disabilitazione dell'interpretazione dei metacaratteri
- Uso di caratteri per disabilitare l'interpretazione di metacaratteri:
 - Backslash: quota il carattere che segue (esempio \#)
 - Single quote: tutti i caratteri circondati da ' perdono il loro significato tranne il carattere '
 - Double quote: tutti i caratteri circondati da " perdono il loro significato tranne \$ ` \ "
- Variabili d'ambiente
 - \$set --- mostra tutte le variabili definite fino a questo momento
 - \$env --- mostra solo le variabili d'ambiente definite dal sistema

Variabili

- Nomi: combinazione illimitata di lettere, numeri e underscore, MA:
NON possono cominciare con numeri e sono CASE sensitive
- Lettura delle variabili dal terminale: istruzione read:
 \$read nome
 \$read uno due tre
- Il valore di una variabile si recupera con la scrittura \$nome.
ESEMPIO:
 \$echo "x" # scrive x
 \$echo "\$y"
 \$echo "\$x" # scrive il valore
 \$echo "y=\$x"
 \$echo "\$y"
- Assegnazione: a=100 (senza spazi), a=\$b

Tipi di dati

- USO IMPROPRI DEL SEGNO \$:

Y=50	→ ok
x=y	→ assegna il nome y ad x
x=\$y	→ assegna 50 ad x
\$x=\$y	→ errore
- TIPI DI DATI (definiti con gli statement typeset o declare):
 - Costanti: typeset (o declare) -r nome_della_costante=valore
\$ declare -r a=1
 - Stringhe (default) oppure typeset

```
$typeset lettera, stringa_numerica, messaggio
$declare lettera, stringa_numerica, messaggio
$lettera="a"
$stringa_numerica="12345"
$messaggio="chiamami al 3861"
$echo "ecco qualche stringa"
$echo "$lettera, $stringa_numerica,$messaggio"
```
 - Interi: typeset -i variabile oppure declare -i variabile

```
$typeset -i a
$a=100
$echo "a = $a"
```

Array

- Dichiarazione di array: typeset (o declare) -a nome_array

```
#array di stringhe
$declare -a arr
$arr[0]="primo"
$arr[1]="secondo"
$arr[10]="decimo"
#Array di interi:
typeset -i b
b[1]=1
b[2]=2
```

- lettura di un elemento dell'array: read b[10]

- lettura dell'array fino al CR: read b

- Stampa di un array: #Valori individuali

```
echo "elemento 0 = ${vettore[0]}"
indice=2
echo "elemento $indice = "
${vettore[$indice]}
echo ${a[1]}
```

Operazioni matematiche

- Solo numeri interi con segno
- Se si usano valori floating non segnala errore ma fa i calcoli con numeri interi
- Operazioni ammesse: + - * / % << >> & ^ (or esc.) |
- Operazioni matematiche anche su variabili stringa (conversione implicita)
- Se si racchiude l'operazione tra ((.)) il risultato viene convertito in numero altrimenti resta stringa
- Numeri in base 2, 8, 16

```
$typeset -i x=123
$typeset -i2 y
$typeset -i8 z
$typeset -i16 h
$h=z=y=x # conversione automatica
```

Struttura di controllo if

STATEMENT if-elif-then-else-fi .

```
read n1 n2
if (( n1<n2 ))
then echo "$n1 minore di $n2"
else echo "$n2 minore di $n1"
fi
=====
```

Oppure ((\$n1<\$n2))

Oppure:

```
then
      echo "$n1 minore di $n2"
else
      echo "$n2 minore di $n1"
fi
```

```
read n1 n2
if (( n1<n2 ))
then echo "$n1 minore di $n2"
elif (( n1==n2 ))
then echo "$n1 uguale a $n2"
else echo "$n1 maggiore di $n2"
fi
```

Condizioni e confronti

- TRA NUMERI → (())
- TRA STRINGHE → [[]]
- DUE PUNTI (:) → CONDIZIONE TRUE
- TEST SU NUMERI: == != < > <= >=
- TEST SU STRINGHE: = != > < -z (stringa nulla)
- Esempi:

```
#!/bin/bash
S1='string'
S2='String'
if [[ $S1 != $S2 ]]
then
    echo "$S1 non uguale a $S2"
fi
```

```
#!/bin/bash
echo "scrivi una stringa"
read nome
if [[ $nome = c* ]]
then
    echo "$nome comincia con c"
else
    echo "non comincia con c"
fi
```

Operatori logici

- Operatori logici AND, OR: || &&

```
#!/bin/bash
declare -i x
declare -i y
declare -i z
read x
read y
read z
if ((x<y)) && ((x<z))
then
    echo "$x e' minore di $y e $z"
fi
```

Operatori logici

■ Possibili opzioni :

- a nome → esiste?
- f nome → e' un file regolare?
- d nome → e' un directory?
- c nome → e' un file di caratteri?
- b nome → e' un file a blocchi?
- p nome → e' una pipe?
- S nome → e' un socket?
- L nome → e' un link ad un altro oggetto?
- s nome → e' non vuoto?

```
[[ -a $nome ]] && [[ -f $nome ]] && [[ -s $nome ]]
```

- r nome → posso leggere?
- w nome → posso modificarlo?
- x nome → posso eseguirlo?
- O nome → ne sono proprietario?
- G nome → e' il mio gruppo?

```
[[ -r $nome ]] && [[ -w $nome ]] && [[ -x $nome ]]
```

Strutture di controllo

■ STATEMENT case-esac

```
echo "scrivi nome"
read nome
case $nome in
    nome1) echo "primo caso"
            echo ;;
    nome2) echo "secondo caso"
            echo ;;
    [a-z][a-z]) echo "coppia"
                echo ;;
    *)      echo "caso inatteso"
            ;;
esac
```

```
for file in *
do  # per ogni elemento della dir corrente
    case $file in  # visualizza un messaggio
        *.txt) echo "$file: file di testo" ;;

        *.gif) echo "$file: file grafico" ;;

        *.pdf) echo "$file: file PDF" ;;

        *.c) echo "$file: file sorgente" ;;

        *) echo "$file: file generico" ;;

esac
done
```

Strutture di controllo

■ STATEMENT WHILE

```
declare -i n=0  
while ((n<4))  
do  
((n=n+1))  
done
```

■ STATEMENT UNTIL

```
declare -i n=0  
until((n>4))  
do  
((n=n+1))  
done
```

■ STATEMENT for

```
for n in 1 2 3 4  
do  
echo "valore di n = $n"  
done
```

```
for nome in mario giuseppe vittorio  
do  
echo "$nome"  
done
```

```
typeset -i ris=5  
for n in 10 100 1000  
do  
((ris=$ris*n))  
done  
echo "ris=$ris"
```

oppure

```
typeset -i ris=5  
for n in 10 100 1000  
do ((ris=$ris*n))  
done  
echo "ris=$ris"
```

Qualche esempio

```
for parola in $linea  #parsing di una stringa
do
    echo "$parola"
done

for nome in *      # lista dei file nella directory corrente
do                  # con permesso di lettura
    if [[ -f $nome ]] && [[ -r $nome ]] && [[ -w $nome ]]
    then
        echo " il file regolare $nome puo' essere letto e scritto"
    fi
done
```

- Comandi composti: separati da punto e virgola

```
$for n in *; do if [ -d $n ]; then echo $n; fi; done
$ls -l|while read l;do echo $l;done
```



Attenzione: il ';' va dove ci sarebbe return nella notazione
compatta della slide 8 !

Parametri di linea ad uno script

- Possono essere:
 - Argomenti semplici (numeri stringhe pathname), opzioni (per esempio `-x` o `+x`), redirezioni (`>` o `<`)
- Parametri posizionali:
 - `$$` → PID
 - `$#` → NUMERO DEI PARAMETRI
 - `$*` → STRINGA FORMATA DA TUTTI I PARAMETRI
 - `$@` → ESPANDE GLI ARGOMENTI
 - `$0` → NOME DELLO SCRIPT, DELLA FUNZIONE
 - `$1...$9,${10}...` → PARAMETRI

```
#!/bin/bash
if (($# > 4))
then
    echo "troppi parametri"
elif (($# == 4))
then
    echo "ok"
    for i in $@
    do
        echo      "$i"
    done
else
    echo "$USAGE"
fi
```

Parametri di linea ad uno script

- Funzioni in BASH
- Variabili globali: dichiarate implicitamente o dichiarate fuori da una funzione
- Variabili locali: definite all'interno di una funzione con `typedef` o `declare`
- Le variabili riservate sono globali
- E' ammessa la ricorsione
- I parametri passati ad una funzione sono recuperati con il meccanismo degli script
- Uno script può chiamare un altro script con lo stesso meccanismo del passaggio di parametri
- Ritorno parametri:
 - Con `Return`: ritorno un valore di 8 bit nella variabile `$?` ATTENZIONE: salvare subito `$?` perchè è usata da tutti i comandi
 - tramite variabili globali
 - tramite file
 - con la scrittura `a=$(nome_funzione param)`

Parametri di linea

ESEMPI

```
function sqr
{
    ((s=$1*$1))
    echo "quadrato di $1 = $s"
}
n=5
sqr $n
```

```
function sqr
{
    ((s=$1*$1))
    return $s
}
n=5
sqr $n
p=$?      # da salvare perche' echo modifica la $?
echo "quadrato di $n = $p"
```

Input/Output

- Istruzioni di base:
 - Read
 - Echo
 - Exec → apre e chiude streams
 - operatori per la redirezione
- read var
 - il terminatore dell'ingresso è dato dalla variabile IFS
 - di default, IFS=spazio|tab|return
 - Esempio di ridefinizione di IFS: IFS="."|

```
declare -i var      # Lettura all'interno di un ciclo:  
declare -i tot=0  
while read var      # continua a leggere fino a quando scrivo ^d  
do  
    ((tot=$tot+$var))  
done  
echo "totale = $tot"
```

Input/Output

- Redirezione dell'input:

- Totale: \$script < file

- Parziale:
while read var
do
 ((tot=tot+var))
done < aa
echo \$tot

- REDIREZIONI:

```
#parsing di un file
read inp
read out
while read stringa
do
    for word in $stringa
    do
        echo "$word"
    done
done < $inp > $out
```

Exec

- Istruzione Exec: apre un file per lettura e scrittura

- Apertura di un file per lettura:

```
exec 8< file # 8 e' il descrittore del file
```

- Apertura di un file per scrittura: exec 8> file # idem

- Chiusura di un file : exec 8<&-

- Lettura/scrittura del file: read -u8 var / echo -u8 \$var

- Cattura dell'output (testuale) di un comando: var=\$(comando-Unix)
ESEMPIO:

```
var=$(ls -l)
var=$(sort filename)
```

- Lettura di un file in una variabile stringa:

```
var=< filename
```

Segnali

- Segnali inviata dalla tastiera:

```
control-c → INT (segnale nr.2)  
control-\ → QUIT (segnale nr. 3)  
control-s → STOP (segnale nr. 17)  
control-q → CONT (segnale nr. 19)
```

- Segnali inviati da un processo: comando kill

```
$kill [-nomesegnale| -numerosegnale] PID
```

- Per catturare un segnale (tranne il segnale 9): istruzione TRAP

- Sintassi: trap 'uno o piu' comandi Unix separati da ;' segnale

- ERRORE DI SCRIPT: trap 'echo "c'e' stato un errore"' ERR

- FILE TEMPORANEO: trap 'rm /tmp/* > /dev/null ; exit' EXIT

NB: EXIT è il segnale 0, che un processo invia al kernel quando termina

- Per evitare che un processo sia terminato da tastiera: trap '' INT QUIT

Segnali

- Esempi di cattura segnale

```
#!/bin/bash
# questo script cattura i segnali
trap 'echo "ho ricevuto quit"' QUIT
trap 'echo "ho ricevuto int"' INT
```

```
count=1
while (:) #while true
do
    (( count=$count+1 ))
done
```

Varie

- Lunghezza stringhe: \${#stringa}
- Pattern matching in bash:
 - * corrisponde a tutte le stringhe
 - ? corrisponde ad un singolo carattere qualsiasi
 - \X disinibisce il significato particolare del carattere X.
 - Esempio: \\ rappresenta \
 - [...] corrisponde a uno dei caratteri racchiusi
 - [X-Z] corrisponde a tutti i caratteri da X a Z
 - [^...] (oppure [!...]) corrisponde a tutti i caratteri che non ci sono

Ereditare variabili

- Le variabili d'ambiente sono ereditate dai processi creati
- Le variabili utente NON sono ereditate dai processi creati
- `$export nome_variabile` fa sì che i processi creati ereditino la variabile
- I processi creati NON modificano l'ambiente del padre
- Per far sì che uno script modifichi l'ambiente del padre --> dot script
- Esempio:

```
$ . ./script
```

Dove Script può essere:

```
#!/bin/bash  
cd $1
```

Altri esercizi con riga di comando

- Contare i file creati, ad esempio, il 30 settembre

```
ls -l|while read l;do echo $l|cut -f6-7 -d" ";done|grep "set 30"|wc -l
```

- Contare i processi che appartengono ad un dato proprietario
- Contare i file che appartengono ad un dato proprietario
- Contare i processi creati da init

```
ps -ef|while read l; do case $(echo $l|cut -f3-3 -d" ") in 1)echo 1;esac;done|wc -l
```

Esempi di Script

```
#!/bin/bash
#scrive il nome dei file leggibili e scrivibili se il nome contiene la stringa data
USAGE='Use $0 <string>'
if [ "$#" != "1" ]
then
    echo $USAGE
else
    for file in *
    do
        if [[ $file = *$1* ]] && [[ -r $file ]] && [[ -w $file ]]
        then
            echo $file
        fi
    done
fi
=====
#!/bin/bash
# verifica se la subdir esiste; se esiste, cambia directory corrente,
# se non esiste, crea la directory e cambia directory.
#
if [ "$#" -eq "0" ]
then
    echo "USAGE: $0 nomedir"
    exit
fi

if [[ -a $1 ]]
then
    cd $1
else
    mkdir $1;cd $1
fi
echo "script $0. Sono nella directory $PWD"
```

Esempi di Script

```
#!/bin/bash
# questo script aggiunge una intestazione data ai file specificati
#script "lista file" "intestazione"
for i in $1    #per tutti i file specificati
do
    echo -e -n "#\n#$2\n#\n" > temp
    cat temp $i > templ
    mv templ $i
done
```

```
#!/bin/bash
#scrive la stringa passata come argomento carattere per carattere
declare -i i
declare -i n
l=$1
i=1
while (( i < ${#l} ))
do
    n=$i
    echo $(echo $l | cut -c $n-$n)
    i=$((i+1))
done
```

Esempi di Script

```
#!/bin/bash
#script stringa1 stringa2 stringa3
# se stringa2 e' contenuta in stringa1 sostituisce stringa2 con stringa3
#
a=$1
b=$2
c=$3
l1=${#a}
l2=${#b}
l3=${#c}
i=1
if [[ $a == *$b* ]]
then
    while(( $i<$l1 ))
    do
        (( i1=$i+$l2-1 ))
        c1=$(echo $a|cut -c${i1}-)
        if [[ ${c1} == $b ]]
        then
            ((start=$i))
        fi
        ((i=$i+1))
        done
        (( end=$start+$l2 ))
        (( start=$start-1))
        testa=$(echo $a|cut -c${start}-)
        coda=$(echo $a|cut -c${end}-)
        echo "$testa$coda"
    fi
```

Esempi di Script

```
#!/bin/bash
#script stringa1 stringa2 stringa3
# se stringa2 e' contenuta in stringa1 sostituisce stringa2 con stringa3
#
#
#           Elaborazione di stringhe in bash
#rimpiazza pattern con replacement in string:
#   ${string/pattern/replacement}
#
a=$1
b=$2
c=$3
echo ${a/$b/$c}
```

Esempi di Script

```
#!/bin/bash
# questo script sostituisce le parole che contengono una stringa p1 con la stringa p2
# nel file indicato nella lista data come terzo argomento
# ./subs.sh p1 p2 lista_file
for i in $3    #per tutti i file indicati
do
    echo "file $i"
    while read n  # $n contiene una stringa
    do
        for k in $n  # $k contiene parole
        do
            if [[ $k = *$1* ]]
            then
                echo -n "$2 "
            else
                echo -n "$k "
            fi
        done
        echo
    done < $i > temp
    mv temp $i
done
```

Esempi di Script

```
#!/bin/bash
#scrive le righe del file dal carattere c1 al carattere c2
#
USAGE="Use: $0 <char1> <char2> <file>."
declare flag="OFF"
declare -i c
declare -i length
if (( $# != 3 ))
then    echo $USAGE
elif [[ $1 != ? ]] || [[ $2 != ? ]]
then    echo $USAGE
elif [[ ! -f $3 ]] && [[ ! -r $3 ]]
then    echo "Il file non esiste o non è leggibile"
else
    while read line
    do
        for word in $line
        do
            ((c=1))
            ((length=$(echo $word | wc -c))-1) #uguale a length=${#word}
            while (( $c < length ))
            do
                character=$(echo $word | cut -c $c)
                if [[ $character = ${char2} ]]
                then    flag="OFF"
                fi
                if [[ $flag = "ON" ]]
                then    echo -n $character
                fi
                if [[ $character = ${char1} ]]
                then    flag="ON"
                fi
                ((c=$c+1))
            done
        done
    done < $3
fi
```

Esempi di Script

```
#!/bin/bash
USAGE="myls.ksh, stampa la lista dei file ordinati nel formato size-nome-filemode-data"

echo -e "\n File ordinati per data \n"
echo -e "SIZE-NOME-FILE mode-DATA \n"

ls -latr|while read l;do echo $l|cut -f5-5 -d" ";done >size.tmp
ls -latr|while read l;do echo $l|cut -f9-9 -d" ";done >nome.tmp
ls -latr|while read l;do echo $l|cut -f1-1 -d" ";done >mode.tmp
ls -latr|while read l;do echo $l|cut -f6-7 -d" ";done >data.tmp

paste size.tmp nome.tmp mode.tmp data.tmp
rm *.tmp # rimozione file temporanei

echo -e "\n\n return.. \n\n"
read nn

echo -e "\n File ordinati per dimensione \n"
echo -e "SIZE-NOME-FILE mode-DATA \n"

ls -laSr|while read l;do echo $l|cut -f5-5 -d" ";done >size.tmp
ls -laSr|while read l;do echo $l|cut -f9-9 -d" ";done >nome.tmp
ls -laSr|while read l;do echo $l|cut -f1-1 -d" ";done >mode.tmp
ls -laSr|while read l;do echo $l|cut -f6-7 -d" ";done >data.tmp

paste size.tmp nome.tmp mode.tmp data.tmp
rm *.tmp
```

Esempi di Script

```
#!/bin/bash
#legge tutti i file con estensione .log selezionando solo le righe di commento
#che contengono la parola Giorgio e le scrivono accodandole sul file commenti.log
#Si supponga che le linee di commento inizino con #
#La sua attivazione può essere
#./script >> commenti.ksh
#
for n in *.log
do
    echo "file $n"
    while read linea
    do
        if [[ $linea = "#"* ]]
        then
            for nn in $linea
            do
                if [[ $nn = "Giorgio" ]]
                then
                    echo "$linea"
                fi
            done
        fi
    done < $n
done
```

Esempi di Script

```
#!/bin/bash
#scrive il file dato come argomento per righe alternativamente minuscole e maiuscole
#
USAGE="Use: $0 <file>."
declare -i nlet=0
if (( $# != 1 ))
then
    echo $USAGE
else
    if [[ ! -f $1 ]]
    then
        echo "il File non esiste!"
    else
        while read line
        do
            for word in $line
            do
                if (( $nlet%2 == 0 ))
                then
                    echo -n $word " "
                else
                    echo -n $word " " | tr '[a-z]' '[A-Z]'
                fi
                ((nlet=$nlet+1))
            done
            echo -e -n "\n"
        done < $1
    fi
fi
```

Esempi di Script

```
#!/bin/bash
#Questo script legge un file di testo (in cui nome viene dato in linea) e accoda in un
#secondo file (anch'esso dato in linea) solo le righe che contengono parole che
#cominciano (opzione -d) o finiscono (opzione -c) con un numero.
#
#script [-d][-c] <nome1> <nome2> Il nr. corretto di parametri e' 3
#
USAGE="script [-d][-c] <file1> <file2>"
if (( $# != 3 ))
then
    echo "errore nei parametri!"
    echo "$USAGE"
else
    case $1 in
        -d)      while read linea
                  do
                      for n in $linea
                      do
                          if [[ $n = ["0"-"9"]* ]]
                          then
                              echo "$linea"
                          fi
                      done
                      done < $2 >> $3 ;;
        -c)      while read linea
                  do
                      for n in $linea
                      do
                          if [[ $n = *[0"-"9] ]]
                          then
                              echo "$linea"
                          fi
                      done
                      done < $2 >> $3;;
        *) echo "switch errato!";;
    esac
fi
```