PROGRAMMAZIONE IN KORN SHELL

E.M.

- DAVID KORN, BELL LABS, "THE KORN SHELL COMMAND AND PROGRAMMING LANGUAGE", 89, PRENTICE HALL

PRIMO ESEMPIO: SCRIPT FILE "primo" -> echo $RANDOM

- ESECUZIONE: → DALLA KORN-SHELL (PROMPT $) → $primo

IL FILE DEVE AVERE LE PROTEZIONI DI LETTURA ED ESECUZIONE. ALTRIMENTI: $chmod +rx primo

→ OPPURE, CHIAMANDO LA SHELL ESPLICITAMENTE: $ksh primo
BASTA IL PERMESSO DI LETTURA

- COMMENTI NELLO SCRIPT: CARATTERE #

SI POSSONO METTERE ANCHE RIGHE VUOTE.

- ATTENZIONE!! IL NOME DELLO SCRIPT: PUO' ESSERE UN NOME RISERVATO (SHELL STATEMENT O COMANDO UNIX). → script.ksh

- DEBUG

- DEBUG TOTALE:

$ksh –x scriptfile

- DEBUG PARZIALE

NELLO SCRIPT : …

```
set –x          # START DEBUG
```

```
…
set +x # STOP DEBUG
```

- LINEA USAGE: USAGE = “usage: script parameteri”

NON VIENE ESEGUITA, MA PUO’ ESSERE STAMPATA CON
grep USAGE nome_script

- SCRITTURE DA SCRIPT

- echo
- print (PIU’ USATA IN KORN)

ESEMPI DI PRINT

$print ciao a tutti
$print “ciao a tutti”
$print –n “ciao” → INIBISCE EOL
$print –n “a tuttti”
$print
$print “\nciao \na tutti” → \n introduce una newline
$print “\t ciao a tutti” → \t e’ un TAB

```
#!/usr/bin/ksh
#nome dello script: s1.ksh
#Questa prima riga serve ad indicare l’interprete usato da questo script.
#Se la shell corrente e’ la Korn, questa riga e’ inutile.
#Ma, se non e’ la Korn, e’ l’unico modo di eseguire la shell come comando
#altrimenti dovrei scrivere $ksh s1.ksh
#NB: s1.ksh deve comunque essere abilitato alla scrittura
print Questa e\’ una prima prova
```

- USO DEGLI APOSTROFI ’ “

CARATTERI CON SIGNIFICATO PARTICOLARE: $\#?[]*{}|();”’

- APOSTROFO SINGOLO: QUELLO CHE E’ RACCHIUSO TRA APOSTROFI SINGOLI PERDE IL SUO SIGNIFICATO PARTICOLARE

ESEMPIO
```
$print ‘$x’      → SCRIVE $x
$print “$x”      → SCRIVE IL VALORE DI x
```

OPPURE: NELLA CANCELLAZIONE:
```
$rm nome     SE nome HA CARATTERI SPECIALI, P.ES. $, ALLORA: $rm ‘$nome’
```

- BACKSLASH: QUELLO CHE SEGUE IMMEDIATAMENTE PERDE IL SUO SIGNIFICATO PARTICOLARE

ESEMPIO:
```
$rm \$nome
$rm ‘$’nome
```

COME SI STAMPANO?

```
$print ‘ciao ‘a’ tutti’       → SCRIVE    ciao ‘a’ tutti
$print ciao \’a\’ tutti       → SCRIVE    ciao ‘a’ tutti
$print ‘ciao “a” tutti’       → SCRIVE    ciao “a” tutti
```

CONTENUTO DELLA DIRECTORY: CARATTERE *:
#conta il numero di file nella mia directory corrente

```
print *|wc -w
```

VARIABILI

- NOMI: COMBINAZIONI DI LETTERE, NUMERI E CARATTERI SPECIALI MA NON POSSONO COMINCIARE CON UN NUMERO. IL VALORE DI UNA VARIABILE E' ALFABETICO
- LUNGHEZZA: NESSUN LIMITE
- QUANTITA' DI VARIABILI: ILLIMITATE
- LISTA DELLE VARIABILI: $set
- CASE SENSITIVE
- ASSEGNAZIONE

```
1 - n=100    (SENZA SPAZI AI LATI DI = PERCHE' ALFABETICO!!)
2 - let n=100
3 – let "n = 100"
4 – ((n = 100))
```

- VALORE DI UNA VARIABILE nome: SINTASSI $nome
- ASSEGNAZIONE DI TRA DUE VARIABILI: x=$n
- SCRITTURA DI UNA VARIABILE: print "$x"

ESEMPIO:

```
$print "x"        → scrive x
$print "$x"       → scrive il valore di x
$print "y=$x"     → assegna x a y
$print "$y"       → scrive il valore di y
$print "$USAGE="      #ATTENZIONE AGLI SPAZI!
```

- INPUT DA SCRIPT: CON L'ISTRUZIONE read:

```
ingsun2/home/mumolo $ read nome
pippo
ingsun2/home/mumolo $ print $nome
pippo
ingsun2/home/mumolo $ read uno due tre
pippo pluto topolino
ingsun2/home/mumolo $ print $uno $due $tre
pippo pluto topolino
```

USO IMPROPRIO DEL SEGNO $:

```
y= 50            → errore
y=" 50"          → ok
y=50             → ok
x=y              → assegna il nome y ad x
x=$y             → assegna 50 ad x
$x=$y            → errore
```

- TIPI DI DATI

- COSTANTI
- STRINGHE (DEFAULT)
- INTERI
- ARRAY
- GLOBALI (DEFAULT)
- STATEMENT typeset PER DEFINIRE I TIPI DI VARIABILI
- DIMENSIONE ARRAY: DIPENDE DALLA VERSIONE DELLA SHELL, MA TIPICAMENTE 512 O 1024
- ARRAY: SOLO AD UNA DIMENSIONE

- DICHIARAZIONE DI COSTANTI: `typeset -r nome_della_costante=valore`

- DICHIARAZIONE DI STRINGHE: LE STRINGHE VENGONO DEFINITE NON APPENA VIENE SCRITTO IL NOME DELLA VARIABILE, ALTRIMENTI, SI PUO' USARE LO STATEMENT `typeset`

  ESEMPIO:

```
$typeset lettera stringa_numerica messaggio
$lettera="a"
$stringa_numerica="12345"
$messaggio="chiamami al 3861"
$print "ecco qualche stringa"
$print "$lettera, $stringa_numerica, $messaggio"
```

- DICHIARAZIONE DI VARIABILI INTERE: SI PUO' USARE LO STATEMENT `typeset -i` OPPURE
  `integer`
  ESEMPIO:
  ```
  $typeset -i variabile
  $integer variabile
  ESEMPIO:
  $integer a
  $a=100
  $print "a = $a"
  ```

- INIZIALIZZAZIONE DI VARIABILI: `integer b=100`

- DICHIARAZIONE DI ARRAY: GLI ARRAY VENGONO DEFINITI APPENA VIENE ASSEGNATO UN VALORE ALL'ARRAY.

  ESEMPIO DI ARRAY DI STRINGHE:
  ```
  $a[0]="primo"
  $a[1]="secondo"
  $a[10]="decimo"
  ```

NB: QUESTO E' UN ARRAY DI STRINGHE PERCHE' NON E' DICHIARATO DIVERSAMENTE. GLI ELEMENTI NON DICHIARATI SONO STRINGHE NULLE.

ESEMPIO DI ARRAY DI INTERI:
```
$integer b
$b[1]=1
$b[2]=2

$read b[10]
```

- INIZIALIZZAZIONE DI UN ARRAY a:
  ```
  $set -A a primo secondo terzo
  $ingsun2/home/mumolo $ print ${a[*]}
  $primo secondo terzo
  ```

- STAMPA DI UN ARRAY:

  - VALORI INDIVIDUALI

```
print "elemento 0 = ${vettore[0]}
indice=2
print "elemento $indice = ${vettore[$indice]}
print
```

  - TUTTO L'ARRAY

```
print "array intero = ${vettore[*]}
print
```

  - L'USO DELLE {} VA FATTO SOLO QUANDO SI PRELEVA UN VALORE. ESEMPIO:

```
a[1]="primo"
print ${a[1]}
v[0]=${a[1]}
```

OPERAZIONI MATEMATICHE

- SOLO NUMERI INTERI (32 BIT) CON SEGNO. NON SEGNALA ERRORE SE SI USANO VARIABILI FLOATING POINT MA ESEGUE OPERAZIONI INTERE!

- OPERAZIONI AMMESSE:

+ - * / % << >> & ^(or esc.) |

- SINTASSI CON DOPPIE PARENTESI! ESEMPIO:

```
integer x
integer y
integer z

((z=x+y))
((z=x-y))
((z=x*y))
print " numero complessivo = $((x+y+z))"
```

- OPERAZIONI MATEMATICHE ANCHE SU VARIABILI DI TIPO STRINGA!

- STAMPA DI NUMERI NEGATIVI: OPZIONE -R (ALTRIMENTI IL SEGNO MENO (-) E' CONSIDERATO UNA OPZIONE!)

- PRIORITA' DELLE OPERAZIONI:

  1) MOLTIPLICAZIONI
  2) DIVISIONI
  3) SOMME E SOTTRAZIONI

- NUMERI IN BASE 2, 8, 16: `typeset –i2, -i8, -i16`

```
typeset -i x=123
typeset -i2 y
typeset -i8 z
typeset -i16 h

h=z=y=x      ( CONVERSIONE AUTOMATICA )
```

- LETTURA DA TASTIERA:

SCRIVERE 2#1000 O 8#12 O 16#56

- NUMERO DI CIFRE:

```
typeset -Zn variabile ( USA n CIFRE. SOLO CON STRINGHE )
```

ESEMPIO:

```
typeset -Z4 numero
numero=12345
print $numero    ( 2345 )
numero=123
print $numero    ( 0123 )
```

```
#!/usr/bin/ksh
#inverte gli elementi di un array
a[0]=1
a[1]=2
a[2]=3
a[3]=4
i=0
n=3
((m=n/2))
while ((i<=m))
do
        temp=${a[$i]}
        print "temp= $temp"
        a[$i]=${a[$n-$i]}
        a[$n-$i]=$temp
        ((i=i+1))
done
print "${a[*]}"
```

PATTERN MATCHING IN KORN SHELL

- USO DI ‘WILDCARDS’. DOVE SI POSSONO USARE LE WILDCARDS?

  1) IN ALCUNI COMANDI UNIX
  2) NELLO STATEMENT CASE
  3) NELLO STATEMENT FOR
  4) NELLO STATEMENT SET
  5) NELLE ISTRUZIONI DI MANIPOLAZIONE STRINGHE

- WILDCARDS:

1) ? → MATCHING CON 1 SOLO CARATTERE

2) $[c_1c_2...c_n]$ → 1 carattere di quelli specificati

   ESEMPIO:
   ca[abz] → caa, cab, caz
   ca[a-z][a-z]
   ca[0-9]
   ca[!abz]
   ca[\-\]\!\\] → ca-, ca], ca! ca\

3) * → CORRISPONDE CON TUTTE LE STRINGHE

4) ?(stringa$_1$| stringa$_2$| … | stringa$_n$) → CORRISPONDE CON NESSUNA O CON UNA DELLE STRINGHE SPECIFICATE. ESEMPIO:

care?(ful|less|free) → care, careful, careless, carefree
care?([a-z]|less) → care, caret, careless
care?(?|??) → care, carez, careto

5) @(stringa$_1$|stringa$_1$|stringa$_3$) → SOLO UNA DELLE STRINGHE.
ESEMPIO:

care@(a|b|c) → carea, careb, carec

6) *(stringa$_1$|...|stringa$_n$) → QUALSIASI STRINGA INCLUSA LA STRINGA NULLA.
ESEMPIO:

P*(A|B) → P, PA, PB, PAB, PAA, ...

7) +(stringa$_1$|...|stringa$_n$) → COME *, MA SENZA LA STRINGA NULLA.

8) !(stringa$_1$|...|stringa$_n$) → CORRISPONDE A TUTTE LE STRINGHE FUORCHE' QUELLE SPECIFICATE
ESEMPIO:

care!(*.bak|*.out) → TUTTE LE STRINGHE TRANNE I NOMI CHE FINISCONO IN .bak O .out

CONDIZIONI E CONFRONTI

- TRA NUMERI → (( ))
- TRA STRINGHE → [[ ]]
- DUE PUNTI (:) → CONDIZIONE TRUE
- TEST SU NUMERI: == != < > <= >=
- TEST SU STRINGHE: = != > < -z (stringa nulla)
- **STATEMENT if-then-else** .

1° ESEMPIO :

```
read n1 n2
if ((n1<n2))
then
    print “$n1 minore di $n2”
else
    print “$n2 minore di $n1”
fi
```

2° ESEMPIO:

```
read n1 n2
if ((n1<n2))
then
    print “$n1 minore di $n2”
elif ((n1==n2))
then
    print “$n1 uguale a $n2”
else
    print “$n1 maggiore di $n2”
fi
```

```
#!/usr/bin/ksh
#trova il minimo e il Massimo di un array
i=0
while ((i<1000))
do
  a[$i]=$RANDOM
  ((i=i+1))
done
#
min=max=${a[0]}
i=1
while ((i<1000))
do
  if ((${a[$i]} < min))
  then
    min=${a[$i]}
  elif ((${a[$i]} > max))
  then
    max=${a[$i]}
  fi
  ((i=i+1))
done
print "massimo=$max, minimo=$min"
```

CONFRONTO TRA STRINGHE

- ATTENZIONE: NON METTERE SPAZI!

ESEMPIO:

```
if [[$x=$y]]
then
  …
```

- USO DEL PATTERN MATCHING

ESEMPIO:

```
print –n “scrivi una stringa”
read nome
if [[$nome=c*]]
then
    print “$nome comincia con c”
fi

if [[$nome!=*c]]
then
    print “$nome non finisce con c”
fi

if [[$nome=@[nome1|nome2|…|nomen]]]
then
    print “$nome e’ uno dei nomi dati”
fi
```

- OPERATORI LOGICI AND, OR: || &&

ESEMPIO:

```
if ((x<y)) && ((x<z))
then
    print “$x e’ minore di $y e $z”
fi
```

- **STATEMENT case-esac**

SINTASSI:

```
case $nome in

    “nome1”) print “primo caso”
             print   ;;

    “nome2”) print “secondo caso”
             print   ;;

    [a-z][a-z])  print “coppia di caratteri”
             print   ;;

    str1@[str2|str3)   print “str1str2 oppure str1str3”
             print   ;;

    str1|str2|str3)    print “str1 oppure str2 oppure str3”
                 print;;

    *)       print “caso inatteso” ;;

esac
```

TEST SUI FILE

- ELENCO DELLE OPZIONI (FILE MODE) :

-a nome → esiste?
-f nome → e' un file regolare?
-d nome → e' un direttorio?
-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?

- ELENCO DELLE OPZIONI (PROTEZIONI):

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

- CONFRONTO SULLE DATE:

nome1 -nt nome2 → nome1 piu' nuovo di (newer than) nome2
nome1 -ot nome2 → nome1 piu' vecchio di (older than) nome2

NB: - SE UN FILE NON ESISTE, LA SHELL CONSIDERA IL FILE ESISTENTE PIU' NUOVO

- SE NESSUN FILE ESISTE, IL RISULTATO E’ FALSO

CICLI IN KORN SHELL

- **STATEMENT while**

```
integer n=0
while ((n<4))
do
    ((n=n+1))
done
```

- **STATEMENT until**

```
integer n=0
until((n>4))
do
    ((n=n+1))
done
```

- **STATEMENT for**

```
for n in 1 2 3 4
do
    print “valore di n = $n”
done

integer ris=5
for n in 10 100 1000
do
    ((ris=ris*n))
done
print “ris=$ris”
```

```
for nome in mario giuseppe vittorio
do
    print “$nome”
done
```

- STATEMENTS break E continue: ANALOGO AL LINGUAGGIO C. break ESCE DAL CICLO E continue LO RIPRENDE

- ALTRI ESEMPI: PARSING DI UNA STRINGA IN PAROLE:

```
for parola in $linea
do
    print “$parola”
done
```

- LISTA DEI FILE CON PERMESSO DI LETTURA

```
for nome in *    ← tutti gli oggetti nella directory
do
    if [[-f $nome]] && [[-r $nome]] && [[-w $nome]]
    then
        print “ il file regolare $nome puo’ essere letto e scritto”
    fi
done
```

- LISTA DELLE DIRECTORIES

```
for n in *
do
   if [[ -d $n ]]
   then
        print $n
   fi
done
```

PARAMETRI DI LINEA AD UNO SCRIPT

- POSSONO ESSERE:

- ARGOMENTI SEMPLICI (NUMERI STRINGHE PATHNAME)
- OPZIONI (PER ES. –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

- ESEMPIO D'USO DI $#

```
USAGE="usage: $0 parametri"
if (($# > 4))
then
    print "troppi parametri"
elif(($# == 4))
then
    print "ok"
    for i in $@
    do
        print "$i"
    done
else
    print "$USAGE"
fi
```

- **STATEMENTS set, shift**

```
set par 1 par2 par3 …     ←   assegna valori ai parametri posizionali
set –s                    ←   ordina i parametri
set --                    ←   cancella i parametri
shift n  (es. shift 1   shift 2   … )  ← shift a sinistra la lista dei
parametri
```

- **STATEMENT getopts** → analizza una lista di switch (opzioni), uno alla volta.

CONCETTO:

```
while getopts ab var
do
    case $var in
    a)  …;
    b)  …;
    esac
done
```

USO DEL CARATTERE DUE PUNTI (:) → se messo prima della opzione, invia il carattere ? se lo switch e' invalido

→ se messo dopo l'opzione, mette l'argomento in OPTARG

ESEMPIO:

```
while getopts :x:y: var
do
    case $var in
    x)  print "$OPTARG";;      # opzione –x
    +x) print "$OPTARG";;      # opzione +x
    y)  print "$OPTARG";;
    -y) print "$OPTARG";;
    \?) print "$OPTARG invalido"
        print "$USAGE";;
    esac
done
```

- **STATEMENT select**

Viene usata per realizzare Menu'. E' un ciclo: ad ogni iterazione legge una variabile.

Per uscire dal ciclo usare break o exit

Per continuare il ciclo, usare continue

La var. d'ambiente PS3 e' usata per il prompt.
Esempio:

```
PS3="scrivere un comando:"
select var in stringa1 stringa2 esci
do
    case $var in
    stringa1)    print "hai battuto stringa1";;
    stringa2)    print "hai battuto stringa2";;
    esci)        print "hai optato per l'uscita"
                 exit;;       # si poteva mettere anche break
    esac
done
```

SOSTITUZIONE DEI COMANDI

- FORMATO: $(comando)

ESEMPIO: a=$(comando)

```
$print "la data attuale e' $(date|cut -d' ' -f1-3) e l'ora $(date|cut -d' ' -f4)"
$print "ci sono attualmente $(who -a|wc -l) e sono sono $(who -a)"
```

- FORMATO ALTERNATIVO (compatibilita' con Bourne): `comando` → accento ‘ ‘

```
$print "ci sono ‘who‘ utenti"
```

- REDIREZIONE DELL'INGRESSO:

```
$a=$(</usr/dict/words)
$print $(print $a|wc -w)      #il file words contiene un dizionario di parole

stessa cosa:
$a=$(cat /usr/dict/words)
$print $(print $a|wc -w)
```

- SOSTITUZIONE DELLA $HOME: CARATTERE ~

~ rimpiazzato con $HOME

~utente rimpiazzato con la home directory di utente

ESEMPIO:

```
$ls ~nomeutente  #attenzione ai permessi
$cd ~nomeutente  #attenzione ai permessi
```

FUNZIONI IN KORN

- VARIABILI GLOBALI: DICHIARATE IMPLICITAMENTE (STRINGHE) O DICHIARATE FUORI DA UNA FUNZIONE

- VARIABILI LOCALI: DEFINITE ALL'INTERNO DI UNA FUNZIONE CON typedef O integer

- LE VARIABILI RISERVATE SONO GLOBALI

- E' AMMESSA LA RICORSIONE

- I PARAMETRI PASSATI AD UNA FUNZIONE SONO RECUPERATI CON IL MECCANISMO DEGLI SCRIPT

- UNO SCRIPT PUO' CHIAMARE UN ALTRO SCRIPT: IL PASSAGGIO PARAMETRI E' LO STESSO DELLE FUNZIONI

- RITORNO PARAMETRI:

- CON RETURN: RITORNO UN VALORE INTERO DI 8 BYTE NELLA VARIABILE $? ATTENZIONE: SALVARE SUBITO $? PERCHE' E' USATA DA TUTTI I PROCESSI (COMANDI, SCRIPT …) PER IL VALORE DI RITORNO
- TRAMITE VARIABILI GLOBALI
- TRAMITE FILE
- CON LA SCRITTURA `a=$(nome_funzione  param)`

ESEMPI

```
function sqr
{
    ((s=$1*$1))
    print “quadrato di $1 = $s”
}

function sqr
{
    ((s=$1*$1))
    return $s
}

n=5
sqr $n
p=$?    # da salvare perche’ print modifica la $?
print “quadrato di $n = $p”
```

- PASSAGGIO DI UN ARRAY AD UNA FUNZIONE: `funct ${array[*]}`

ESEMPIO:

```
function minimo
{
    set –s
    print “minimo = $1”
}
a[0]=50
a[1]=1
…
a[100]=33
minimo ${a[*]}
```

ELABORAZIONE DI STRINGHE

- LUNGHEZZA DI UNA STRINGA: `length=$(#stringa)`
- CONFRONTO DI DUE STRINGHE
- CONFRONTO DI UNA STRINGA ED UN PATTERN
- ASSEGNAZIONE DI UN VALORE AD UNA STRINGA
- CONVERSIONE DI UNA STRINGA IN UPPER O LOWER CASE:

`typeset –u stringa` → converte in upper
`typeset –l stringa` → converte in lower

- GIUSTIFICAZIONE A SINISTRA: `typeset –Ln stringa`
- CONCATENAZIONE
- RIMOZIONE DEI CARATTERI IN UNA STRINGA :
  - A SINISTRA: → USO #
  - A DESTRA → USO %

Esempio: rimuovo due caratteri a sx e dx: `${stringa#??} ${stringa%??}`

```
Nota sulla identificazione delle variabili: notazione ${var} per non confondere il nome.
Esempio:
ingsun2/home/u $ a=ab
ingsun2/home/u $ print $a
ab
ingsun2/home/u $ b=$acd  #vorrei concatenare la var $a con cd, ma la shell cerca la var acd
ingsun2/home/mumolo $ print $b
                        # e quindi nin stampa niente
ingsun2/home/mumolo $ b=${a}cd          #per identificare la var a
ingsun2/home/mumolo $ print $b
abcd
ingsun2/home/mumolo $
```

Sommario dei comandi di elab stringhe

```
${#var}      # lunghezza di var

${var#pattern}   #valore di var senza la piu’ piccola porzione iniziale che
corrisponde a pattern

${var##pattern}  # “                    “       grande

${var%pattern}   #valore di var senza la piu’ piccola porzione finale che
corrisponde a pattern

${var%%pattern}  #”      “          grande


ESEMPI:

ingsun2/home/user $ a=abcabcdefdef
ingsun2/home/user $ print ${#a}
12
ingsun2/home/user $ print ${a#abc}
abcdefdef
ingsun2/home/user $ print ${a%def}
abcabcdef
```

INPUT/OUTPUT

- STATEMENT DI BASE:

- read
- print
- exec → apre e chiude streams
- operatori per la redirezione

- READ VAR

- SE NON E' INDICATA LA VARIABILE, L'INPUT VIENE MESSO IN **REPLY**
- PROMPT: read nome?"scrivi il nome"
- IL TERMINATORE DELL'INGRESSO E' DATO DALLA VARIABILE **IFS**
- DI DEFAULT, IFS=spazio|tab|return
- ESEMPIO DI RIDEFINIZIONE DI IFS: IFS= ".|"
- LETTURA DI UN INGRESSO PIU' LUNGO DI UNA LINEA → \

ESEMPIO:

```
read var
questa\
e'\
una\
stringa
```

- LETTURA ALL'INTERNO DI UN CICLO:

```
while read var        # continua a leggere fino a quando scrivo ^d
do
    ((tot=tot+var))
done
```

```
        print “totale = $tot”
```

- REDIREZIONE DELL’INPUT

TOTALE:
```
$script < file
```
PARZIALE:
```
while read var
do
    ((tot=tot+var))
done < file
```

- PRINT

- OPZIONI:
  - \a → bell
  - \b → sovrascrive
  - -r → inibisce il significato del carattere backslash
  - -- → come –R

- REDIREZIONI:

```
#parsing di un file
read inp?“input file”
read out?“output file”
while read stringa
do
    for word in $stringa
    do
        print “$word”
    done
done < $inp > $out
```

- EXEC. APRE UN FILE PER LETTURA E SCRITTURA
  - APERTURA PER READ: exec 8< file # 8 e’ il descrittore del file
  - APERTURA PER PRINT: exec 8> file # idem
  - CHIUSURA DEL FILE: exec 8<&-

- LETTURA/SCRITTURA: read –u8 var / print –u8 $var

CATTURA DELL'OUTPUT ASCII DI UN COMANDO:

```
var=$(comando-Unix)
ESEMPIO:    var=$(ls –l)
            var=$(sort filename)
```

- LETTURA DI UN TEXT FILE IN UNA VARIABILE STRINGA

```
ESEMPIO:    var=$(< filename)
```

FOREGROUND, BACKGROUND

- BACKGROUND PROCESS: NON RICHIEDE DATI INTERATTIVAMENTE MA SOLO TRAMITE FILE
  ESEMPIO: `$comando < filein > fileout&`

- OPERATORE DI BACKGROUND → &

- CONTROLLO DEI PROCESSI IN BACKGROUND:

```
$bg                 →  porta il processo correntemente stoppato da fg a bg
$fg                 →  muove il processo correntemente in backg in fg
$bg %n              →   "          "     n
$fg %n              →   "          "     n
$kill –STOP processo →  ferma un processo
$kill –CONT processo → riavvia il processo
$kill 0             → manda il segnale 15 a tutti i processi in background
$jobs [-l –p]       →  lista i numeri e gli identificatori dei processi in
                        background.  -l lista lo stato    -p  mostra i PID
$wait [%n]          → aspetta il completamento del processo background n
Ctrl-z              → stop il processo corrente
```

SEGNALI

- SEGNALI DA TASTIERA:

- SEGNALE DA PROCESSO: comando kill
  ```
  $kill [-nomesegnale| -numerosegnale] PID
  ```

- TUTTI I SEGNALI ECCETTO STOP E CONT POSSONO TERMINARE UN PROCESSO (SCRIPT)

- PER CATTURARE UN SEGNALE: STATEMENT TRAP

- SI POSSONO CATTURARE DIVERSI SEGNALI

- NON SI PUO' CATTURARE IL SEGNALE KILL

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

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

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

NB: EXIT E' IL SEGNALE 0, CHE UN PROCESSO INVIA AL KERNEL QUANDO TERMINA

- PER EVITARE CHE UN PROCESSO SIA BLOCCATO DA TASTIERA: `trap "" INT QUIT`

- `trap – segnale` RESETTA IL COMPORTAMENTO ORIGINALE

AMBIENTE

- ALL'AVVIO, SI RICHIAMANO ALCUNI SCRIPT:
  `/etc/password`
  `/etc/profile` → comune a tutti (valore di PATH, valore di TERM …)
  `$HOME/.profile` → del singolo utente

- AMBIENTE: INSIEME DI CARATTERISTICHE, PRIVILEGI, RISORSE

- EREDITARIETA': ALCUNE CARATTERISTICHE VENGONO ESPORATE DAL PADRE AL FIGLIO, MAI VICEVERSA. ESEMPIO:
  SCRIPT cd.ksh:
  ```
  #
  cd $1
  print $PWD
  ```
  → NON MODIFICA LA DIRECTORY DELLA SHELL

- DOT-SCRIPTS: GIRANO NELL'AMBIENTE DEL PADRE! CIOE' PUO' CAMBIARE L'AMBIENTE!!

- NB: LE VAR. RISERVATE, LE VARIABILI E FUNZIONI DEFINITE DAL PADRE NON SONO VISIBILI AL FIGLIO. DEVONO ESSERE ESPORTATE CON `$export var`

- PER VEDERE LE VAR E FUNZIONI ESPORTATE:
  `$typeset –i` → variabili intere
  `$typeset –x` → variabili esportate
  `$typeset –fx` → funzioni esportate

ESEMPIO DI ~/.profile

```
#
function _cd
{
cd "$1"
PS1="$(uname -n)"’$PWD $ ’      #potrebbe essere hostname
}
stty erase ^H
stty istrip
PATH=/usr/bin:/usr/ucb:/etc:.
export PATH
alias dir=’ls’
alias h=’history’
alias cd=_cd
alias pico=’/usr/local/bin/pico’
FCEDIT=vi
export FCEDIT
cd $HOME
#TERM="vt100"
# If possible, start the windows system
#
if [ ‘tty‘ = "/dev/console" -a "$TERM" = "sun" ] ; then
        if [ ${OPENWINHOME:-""} = "" ] ; then
                OPENWINHOME=/usr/openwin
                export OPENWINHOME
        fi
        echo ""
        echo "Starting OpenWindows in 5 seconds (type Control-C to interrupt)"
        sleep 5
        echo ""
        $OPENWINHOME/bin/openwin
        clear           # get rid of annoying cursor rectangle
        exit            # logout after leaving windows system
fi
```

ALIAS

- MACRO DI COMANDI, USATI PER BREVITA'
- FORMATO: `alias nome=valore` ESEMPIO: `$alias p=print`
- SE CI SONO SPAZI BIANCHI: " " Esempio: `$alias l="ls –li"`
- QUALSIASI TESTO O CARATTERI SPECIALI

ESEMPIO: `$alias w="who|wc –l"` #attenzione: sovrascrive il nome del comando w (devo usare path/w)

- ALIAS MULTIPLI: `$alias l="ls –il" w="who|wc –l"`
- `$alias` MOSTRA GLI ALIAS CORRENTI
- `$alias –x` MOSTRA GLI ALIAS ESPORTATI
- `$alias –t` VISUALIZZA LE CORRESPONDENZE TRA NOME E PATH. `$alias –t nome` DEFINISCE UNA CORRISPONDENZA. ESEMPIO:

```
ingsun2/home/utente/scripts $ alias -t
cat=/usr/bin/cat
chmod=/usr/bin/chmod
cp=/usr/bin/cp
...
$alias –t mycat.ksh  #trova il primo file in base al PATH
$alias –t
...
mycat.ksh=/home/mumolo/scripts/mycat.ksh
rm=/usr/bin/rm
```

- `$unalias l` RIMUOVE L'ALIAS

STORIA DELLA SHELL

- i comandi utente sono accodati nel file ~utente/.sh_history

```
$cat ~utente/.sh_history
```

ACCESSO AL FILE con il comando `fc`

```
$fc –l            → lista gli ultimi 16 comandi numerandoli
$fc -ln           → lista gli ultimi 16 comandi senza numerazione
$fc –lr           → lista gli ultimi 16 comandi in ordine inverso
$fc –l n1 n2      → lista i comandi da n1 a n2
$fc –l –20        → lista gli ultimi 20 comandi

$history          → uguale a fc -l
$history n1 n2    → mostra i comandi da n1 a n2
$history –r n1 n2 → in ordine inverso
$history –n       → ultimi n
```

- EDITING DI UN COMANDO PRECEDENTE:

```
$fc  -e editor –n     → edita ed esegue il comando n

$fc –n                → edita il comando n usando l'editor settato in FCEDIT
$FCEDIT=vi
$export FCEDIT

$fc 98        → chiama l'editor vi per modificare la linea 98 ed esegue

$fc           → stessa cosa con l'ultimo comando
```

- RIESECUZIONE DEI COMANDI PRECEDENTI

```
$r                → riesegue l'ultimo comando
$r  cc            → riesegue l'ultimo
```

```
$r 98                  →  riesegue il comando nr. 98
```