Laboratorio di Sistemi Operativi

primavera 2009

BASH:
Bourne Again Shell

(6)

Controllo del flusso

- Il controllo del flusso dà al programmatore la possibilità di scegliere che solo una parte del codice venga eseguita o che un'altra parte venga eseguita più volte o altro.
- I costrutti supportati da bash per il controllo del flusso sono:
  - if/else
  - for
  - while
  - until
  - case
  - select

if/else

```
if condizione then
    comandi
[elif condizione then
    comandi ...]
[else
    comandi]
fi
```

Condizione Booleana:
[□*str1*□=□*str2*□]; □ = spazio

Condizione Comandi:
grep pippo file
dipende dal valore di uscita del comando
0 indica successo e rende vera la condizione
≠ 0 denota un errore

if/else

```
file=$1
word1=$2
word2=$3

if [ $word1 = $word2 ]; then
        echo "Le parole $word1 e $word2 sono uguali"
        exit 1
fi

if grep –q $word1 $file && grep –q $word2 $file
then
        echo $word1 e $word2 sono contenute in $file
fi
exit 0
```

condizioni [...];

- Stringhe
  - *str1* = *str2* stringhe uguali
  - *str1* != *str2* stringhe diverse
  - *str1* < *str2* *str1* precede *str2*
  - *str1* > *str2* *str2* precede *str1*
  - -n *str1* *str1* non e' nulla (ha lung>0)
  - -z *str1* *str1* e' nulla


```
filename=$1
filename=${filename:?"Manca il nome del file"}
howmany=${2:-5}
sort –nr $filename | head -${howmany}
```

```
if [ -z "$1" ]; then
        echo 'uso: stampaprimi filename N'
else
filename=$1
howmany=${2:-5}
sort –nr $filename | head -${howmany}
fi
```


condizioni [...];

- Test di file
  - -d *file* *file* esiste ed e' una directory
  - -e *file* *file* esiste
  - -f *file* *file* esiste ed è un *file* normale
  - -r *file* abbiamo il permesso per lettura
  - -w *file* abbiamo il permesso per scrittura
  - -x *file* abbiamo il permesso per esecuzione
  - -s *file* *file* esiste e non e' vuoto
  - -O *file* siamo il proprietario del *file*


```
if [ ! -e $1 ]; then
    echo "file $1 does not exist."
    exit 1
fi
if [ -d $1 ]; then
    echo -n "$1 is a directory that you may "
    if [ ! -x $1 ]; then
        echo -n "not "
    fi
    echo "search."
elif [ -f $1 ]; then
    echo "$1 is a regular file."
else
    echo "$1 is a special type of file."
fi
```

```
if [ -O $1 ]; then
    echo 'you own the file.'
else
    echo 'you do not own the file.'
fi

if [ -r $1 ]; then
    echo 'you have read permission on the file.'
fi

if [ -w $1 ]; then
    echo 'you have write permission on the file.'
fi
```


condizioni [...];

- Espressioni aritmetiche
  - *expr1* -eq *expr2*
  - *expr1* -ne *expr2*
  - *expr1* -lt *expr2*
  - *expr1* -gt *expr2*
  - *expr1* -le *expr2*
  - *expr1* -ge *expr2*
- AND e OR: && e || oppure –a e -o
  - Verificare la correttezza:
    - if [$i < 5 -o $j > 4] && [$s1 = $s2]; then
    - if [-n $dir] && [\(-d $dir \) –a \(-x $dir \)]; then
    - if [-e $1] && grep pippo $1 then

for

for *var* in *lista*
do
  comandi che usano $var
done

E' diverso da
for i=0 to 10 oppure for (i=0;i<10;i++)

Cicla su tutti i valori nella lista, ad ogni iterazione la variabile *var* assume uno dei valori

- se in *lista* è omesso, la lista di default è "$@"


```
bash> for file in file1 file2 file3; do cp $file  \
> $file.backup; done
bash>
```

```
IFS=:

for dir in $PATH
do
    if [ -z "$dir" ]; then dir=.; fi

    if ! [ -e "$dir" ]; then
        echo "$dir non esiste"
    elif ! [ -d "$dir" ]; then
        echo "$dir non è una directory"
    else
        ls -ld $dir
    fi
done
```

case

```
case expr in
   pattern1)
        comandi1;;
   pattern2)
        comandi2;;
   ...
esac
```

Se *expr* coincide con uno dei *pattern* allora i corrispondenti *comandi* sono eseguiti

Se nessun pattern che coincide è trovato non accade niente

I pattern sono controllati nell'ordine


```
for nome in "$@"; do
      case $nome in
      *.gif)
            echo $nome e\' un file GIF.
      ;;
      *.jpg)
            echo $nome e\' un file JPG.
      ;;
      *)
            echo Non conosco il formato del file $nome
         ;;
      esac
done
```


while/unitl

```
while condition; do
        comandi
done
```

```
until comando; do
        comandi
done
```

Nel while il loop viene eseguito quando la condizione è vera

Nell' until il loop viene eseguito quando la condizione è falsa

- La condizione è controllata all'inizio del loop
- Scegliendo di mettere un comando al posto della condizione in until ci permette di "Esegui i *comandi* fino a che *comando* gira correttamente"


```
path="$PATH:"
while [ $path ]; do
        ls -ld ${path%%:*}
        path=${path#*:}
done
```

```
until cp $1 $2; do
        echo "Tentativo di copa di $1 in $2 fallito. Attesa ..."
        sleep 5;
done
echo "Copia riuscita"
```

select: input dall'utente

```
select var in lista;
do
        comandi che usano $var
done
```

Tale costrutto non ha analoghi nei linguaggi di programmazione convenzionali

Sintatticamente somiglia a for e come esso può omettere in *lista* che per default è "$@"

select permette di generare facilmente dei semplici menù

- genera un menù in cui sono elencati e numerati gli oggetti presenti in *lista*
- aspetta che l'utente scelga e conserva tale scelta in *var*
- esegue i *comandi*
- continua per sempre e si ferma se tra i comandi da eseguire c'è *break*


```
PS3="quale?"
select scelta in "scelta_num_1" "scelta_num_2"; do
        if [ $scelta ]; then
                echo Hai scelto $scelta; break
        else
                echo Scelta non valida
        fi
done
```

```
bash> ./menu
1) scelta_num_1
2) Scelta_num_2
quale?
```