Laboratorio di Sistemi Operativi

primavera 2009

BASH:
Bourne Again Shell

(5)

Script

- Uno script e' un file con comandi shell

```
!#/bin/bash
echo Questo e' un script
echo I file in /bin sono:
ls /bin
```

- Per eseguirlo
  - source *filename*
  - oppure
    - chmod +x *filename*
    - *./filename*
  - Il secondo metodo lancia una subshell ed esegue lo script nella subshell
- Prima riga !*#comando*, opzionale
  - viene usato *comando* per interpretare lo script

Funzioni di bash

- Lo stesso concetto dei linguaggi di programmazione

```
function nome {
      comandi shell
}

nome () {
      comandi shell
}
```

- Possono essere definite anche fuori dagli script con due vantaggi:
  - Modulare lunghi programmi shell
  - Costruire funzioni utilizzabili da più programmi shell
- declare –f da la lista delle funzioni definite

Parametri

- Quando gli script vengono invocati possono richiedere dei parametri
  - I parametri sono passati in variabili speciali: 0,1,2,....
- $1, $2, ..., ${10}, ${11}, ...
  - $0 e' il nome dello script
- $* e' una stringa contente tutti i parametri separati con il primo carattere contenuto nella variabile IFS
- $@ = "$1" "$2" ... "${N}"
  - cioè N stringhe separate da spazio
- $# = numero di parametri
  - ${$#} ultimo parametro
- Tali parametri sono a sola lettura; il loro valore non può essere modificato nello script

```
function funzione1 { # questa funzione richiede due parametri
        echo funzione $0: $1 $2
        pippo="ciao dalla funzione1"
        echo pippo = $pippo
}
pippo="ciao dallo script"
echo pippo = $pippo
echo script $0: $1 $2
funzione1 parametro1 parametro2
echo sono tornato nello script
echo pippo = $pippo
echo script $0: $1 $2
```

```
bash> ./scriptfile arg1 arg2
pippo = ciao dallo script
script ./scriptfile: arg1 arg2
funzione ./scriptfile: parametro1 parametro2
pippo = ciao dalla funzione1
sono tornato nello script
pippo = ciao dalla funzione1
script ./scriptfile: arg1 arg2
```


Variabili locali e globali

- In uno script:
  - Le variabili sono sempre globali, anche se definite nelle funzioni
  - Si puo' definirle locali esplicitamente
    - all'inizio della funzione
    - local pippo


```
function funzione1 { # questa funzione richiede due parametri
        local pippo
        echo funzione $0: $1 $2
        pippo="ciao dalla funzione1"
        echo pippo = $pippo
}
pippo="ciao dallo script"
echo pippo = $pippo
echo script $0: $1 $2
funzione1 parametro1 parametro2
echo sono tornato nello script
echo pippo = $pippo
echo script $0: $1 $2
```

```
bash> ./scriptfile arg1 arg2
pippo = ciao dallo script
script ./scriptfile: arg1 arg2
funzione ./scriptfile: parametro1 parametro2
pippo = ciao dalla funzione1
sono tornato nello script
pippo = ciao dallo script
script ./scriptfile: arg1 arg2
```


Operatori di stringhe

- Sono molto utili per manipolare valori di variabili senza la necessità di complicati programmi
  - soprattutto per manipolare nomi di file
- Cambiare il nome da .gif a .jpg
  - *nuovonome*=${*vecchionome*%.gif}.jpg
- NOTA: spazi
  - ${*var*}
  - ${*var* : *offset*}
  - vanno scritti senza spazi bianchi, anche se in queste slide sembra che ci siano degli spazi fra alcuni caratteri

Operatori di stringhe

- ${*var* :-*stringa*}
  - se *var* esiste e non e' nulla restituisce ${var}, altrimenti restituisce *stringa*
  - Es: `${count:-0}` se count non è def. mi dà **0**
- ${*var* :=*stringa*}
  - se *var* esiste e non e' nulla restituisce ${var}, altrimenti restituisce *stringa* e *stringa* viene assegnata a *var*
  - Es: `${count:=0}` se count non è def. setta count a 0
- ${*var* :?*messaggio*}
  - se *var* esiste e non e' nulla vale ${var}, altrimenti viene stampato il messaggio e lo script termina
  - Es: `${count:?"non definito"}` se count non è def. stampa **non definito**

Operatori di stringhe

- ${*var* :+*stringa*}
  - se *var* esiste non e' nulla restituisce *stringa*, altrimenti vale NULL
  - Es: `${count:+"true"}` dà **true** se count esiste
- ${*var* : *offset* }
  - sottostringa di ${*var*} inizia alla posizione *offset*.
  - Es: `${count:4}` se count=rescigno dà **igno**

Operatori di stringhe

- ${*var* : *offset* : *len*}
  - sottostringa di ${*var*} di *len* caratteri, da *offset*.
  - Es: `${count:4:2}` se count=rescigno dà **ig**
- ${#var}
  - lunghezza della stringa contenuta in var
  - Es: `${#count}` se count=rescigno dà **8**

Operatori di stringhe

- ${*var* # *pattern*}
  - Se *var* inizia con *pattern*, cancella il match piu' corto e ritorna la rimanente parte
- ${*var* ## *pattern*}
  - Se *var* inizia con *pattern*, cancella il match piu' lungo e ritorna la rimanente parte

```
bash> var=/usr/home/rescigno/pippo.file.txt
bash>
bash> echo ${var#/*/}
home/rescigno/pippo.file.txt
bash> echo ${var##/*/}
pippo.file.txt
bash > echo ${var}
/usr/home/rescigno/pippo.file.txt
bash>
```

Operatori di stringhe

- ${*var* % *pattern*}
  - Se *var* finisce con *pattern*, cancella il match piu' corto e ritorna la rimanente parte
- ${*var* %% *pattern*}
  - Se *var* finisce con *pattern*, cancella il match piu' lungo e ritorna la rimanente parte

```
bash> var=/usr/home/rescigno/pippo.file.txt
bash>
bash> echo ${var%.*}
/usr/home/rescigno/pippo.file
bash> echo ${var%%.*}
/usr/home/rescigno/pippo
bash > echo ${var}
/usr/home/rescigno/pippo.file.txt
bash>
```


Esempio

- Supponiamo di avere un file dati:

```
7    Pino Daniele
12   Battisti
1    De Gregori
2    Zucchero
5    Bennato
...        ....
```

- Vogliamo scrivere uno script che stampa le prime *n* linee dopo aver ordinato i dati in ordine di numero di CD decrescente
- Lo script prende in input il nome del file ed il valore *n* (opzionale)

Esempio

```
# Sintassi:      stampaprimi file [n]
#        stampa le prime n linee del file ordinato per valori decrescenti
#
filename=$1
filename=${filename:?"Manca il nome del file"}
howmany=${2:-5}
sort –nr $filename | head -${howmany}
```

```
bash> chmod u+x stampaprimi
bash> ./stampaprimi dati 3
12  Battisti
7   Pino Daniele
5   Bennato
bash> ./stampaprimi
stampaprimi: filename: Manca il nome del file
bash> ./stampaprimi 3
No such file
bash>
```


Sostituzione comandi

- Permette di usare lo standard output di un programma come se fosse una variabile
- $(comando), es., $(ls $HOME)
- primo=$(stampaprimi $file 1)
- Backtick: `*comando*` equivalente a $(*comando*)
  - obsoleto