Introduzione alla programmazione nella shell di Unix

10 Maggio 2005

1

Shell e programmazione

- La shell di UNIX definisce un vero e proprio linguaggio di programmazione, le cui caratteristiche principali sono:
  - Uso di variabili
  - Istruzioni di controllo di flusso: if, for, while
  - Procedure con passaggio dei parametri di ingresso

2

Programmazione

- **E` possibile memorizzare sequenze di comandi all'interno di file eseguibili:**

file comandi (script)

Ad esempio:

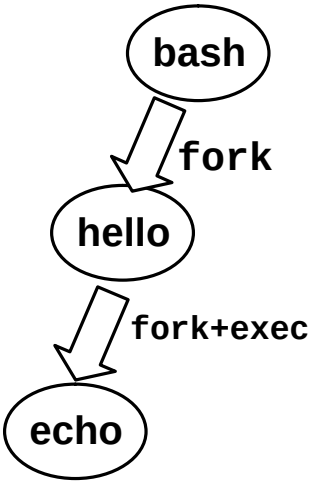
```
#!bin/bash

echo hello world!
```

file hello

```
bash-2.05$ hello

hello world!
```



3

File comandi (*script*)

- **File di testo contenente una serie di comandi**
- **Ha due caratteristiche:**
  - **Deve essere eseguibile**
  - **Deve cominciare con `#!bin/bash`**
    - # è visto dalla shell come commento **ma**
    - **#! è visto dal sistema operativo come *magic number* che identifica uno script; il s.o. capisce così che deve invocare** /bin/bash **come interprete dello script**
- **NOTA BENE: uno script viene** interpretato **(non esiste una fase di compilazione)!**
- **L'interprete acquisisce ed esegue i comandi uno per volta**

4

Ambiente di esecuzione

- **Ogni comando esegue nell'ambiente associato allo shell che esegue il comando**
- **ogni shell eredita l'ambiente dalla shell che l'ha creata**
- **L'ambiente e` costituito da un insieme di variabili (rappresentate da *stringhe*) alle quali il comando puo` fare riferimento:**
  - **variabili con significato standard: PATH, USER, TERM, ecc.)**
  - **variabili user defined**

5

Variabili

Strumenti per gestire le variabili:

- **definizione/assegnamento di variabili:**
  `bash-2.05$ NEWVAR=hello`
  ➔ assegna alla variabile `NEWVAR` la stringa "`hello`"

- **il valore di una variabile si ottiene con la sintassi `$<nomevar>`. Ad esempio:**

```
bash-2.05$ NEWVAR=hello
bash-2.05$ echo $NEWVAR
hello
```

6

Variabili

Per vedere tutte le variabili di ambiente ed i loro valori si puo` utilizzare il comando *set*:

```
bash-2.05$ set
BASH=/usr/bin/bash
HOME=/space/home/wwwlia/www
PATH=/usr/local/bin:/usr/bin:/bin
PPID=7497
PWD=/home/Staff/AnnaCiampolini
SHELL=/usr/bin/bash
TERM=xterm
UID=1015
USER=anna
```

7

Variabili e file comandi

Esistono alcune variabili predefinite che possono essere utilizzate all'interno di file comandi.

- **Per esempio se il file comandi viene invocato con la sintassi:**

```
$ prova arg1 arg2 .. argN
```

  - **$*: e` l'insieme di tutti gli argomenti [arg1, argN]**
  - **$0, $1, ..$N: rappresentano rispettivamente i valori di arg0, arg1,.. argN. (variabili *posizionali*)**
  - **$#: e` il numero degli argomenti passati (escluso arg0)**
  - **$?: e` il valore (stato) restituito dall'ultimo comando eseguito**
  - **$$: e` il pid del processo che esegue il file comandi**

Variabili e file comandi

- **Esempio:**

file comandi prova

```
#!bin/bash
#prova arg1 arg2
echo i miei argomenti: $*
echo il mio direttorio corrente: $PWD
echo il valore del mio pid: $$
echo lo stato dell\'ultimo comando: $?
```

```
bash-2.05$ prova pluto pippo
i miei argomenti: pluto pippo
il mio direttorio corrente: /home/Staff/AnnaCiampolini
il valore del mio pid: 8131
lo stato dell'ultimo comando: 0
```

Espansione

- **Quando un comando contiene uno *schema*, la bash lo interpreta in due passi:**
  1. Sostituzione **dello schema con i nomi che fanno "match"**
  2. Esecuzione **del comando espanso**

  Esempio:

`for i in ../*`
 	
`do`
 	
`  echo $i`
 	
`done`
	1. `../*` viene espanso con la lista dei file contenuti nel direttorio padre
	
	2. viene eseguito il comando vero e proprio, con `i` che cicla sugli elementi della lista espansa

10

Problemi

- **Come faccio a creare un file di nome**
  `( :--> | )` ?
  Devo poter comunicare alla shell che i caratteri speciali vanno privati del loro significato e considerati come normali.

- **Come faccio a dire che nel comando**
  `echo direttorio corrente: pwd`
  `pwd` è esso stesso un comando da eseguire?

- **…**

11

Quoting

\	Il carattere successivo non viene interpetato come carattere speciale
'<testo>'	<testo> viene protetto da qualsiasi tipo di espansione
‘<testo>‘	<testo> viene interpretato come un comando da eseguire
"<testo>"	<testo> viene protetto da espansioni, con eccezione di • ‘ • $ • \

12

Problemi (risolti)

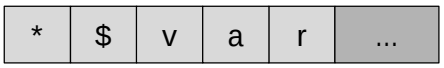
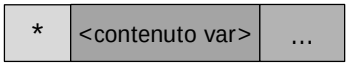
- **Come faccio a creare un file di nome**
  `( :--> | )` ?

  `> \(\ :--\>\ \|\ \)` oppure
  `> "( :--> | )"` oppure
  `> '( :--> | )'`

- **`echo direttorio corrente: ‘pwd‘`**

13

Altri esempi

`rm '*$var'*`	Rimuove i file che fanno match con 
`rm "*$var"*`	Rimuove i file che fanno match con se var non è definita  altrimenti
`echo '<‘pwd‘>'`	Mostra `<‘pwd‘>`
`echo "<‘pwd‘>"`	Mostra *<direttorio corrente>*

14

Espressioni

- **Le variabili sono delle stringhe. Tuttavia e` possibile forzare l'interpretazione numerica di stringhe che contengono la codifica di valori numerici.**
  - **E` possibile eseguire espressioni aritmetiche con il comando `expr` :**

```
expr 1 + 3
```

- **Se dispongo di una variabile numerica var,**

  come faccio a dire che nel comando

```
echo risultato: var+1
```

  `var+1` è il risultato della corrispondente espressione?

```
echo risultato: ‘expr $var + 1‘
```

15

Esempio

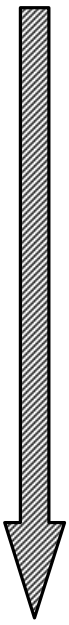
```
#!bin/bash
A=5
B=8
echo A=$A, B=$B
C=`expr $A + $B`
echo C=$C
```

file somma

```
bash-2.05$ somma
A=5, B=8
C=13
```

16

fasi del *parsing* della shell

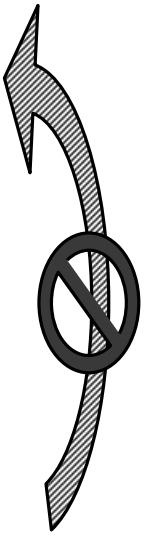


R **ridirezione dell'input / output**
`echo hello > file1 # crea file1 e`
`# collega a file1 lo` ***stdout*** **di** `echo`

1. **sostituzione dei comandi (backquote)**
   `` `pwd` `` ➔ `/temp`

2. **sostituzione di variabili e parametri**
   `$HOME` ➔ `/home/staff/ptorroni`

3. **sostituzione di metacaratteri**
   `comando?` ➔ `comando1 comando2`



17

esempi (parsing)

```
echo `pwd` > “f1>”
# R: crea f1>, poi stdout_echo= f1> ; echo `pwd`
# 1: echo /usr/bin
# 2: echo /usr/bin
# 3: echo /usr/bin

test –f `pwd`/$2 –a –d “$HOME/dir?”
# R: test –f `pwd`/$2 –a –d “$HOME/dir?”
# 1: test –f /temp/$2 –a –d “$HOME/dir?”
# 2: test –f /temp/arg_n2 –a –d \
 “/home/staff/ptorroni/dir?”
# 3: test –f /temp/arg_n2 –a –d \
 /home/staff/ptorroni/dir?
```

18

Comando if

- **Il comando if serve per esprimere l'alternativa tra due (sequenze di) comandi, in base al valore di una condizione logica:**

if <condizione>

then <lista di comandi >

[else <lista di comandi >]

fi

- **le condizioni logiche si possono esprimere mediante il comando `test`:**

```
test $a –eq $b
```

oppure `[$a –eq $b]`

Esempio

file provacond

```
A=5
B=8
echo A=$A, B=$B

if test  `expr $A + $B` -eq 0
then echo VERO!
else echo FALSO!
fi
```

```
bash-2.05$ provacond
A=5, B=8
FALSO!
```

20

Esercizio 1

- **Creare uno script che abbia la sintassi**

  ./ps_monitor.sh [#lines]

- **Lo script,**
  - **in caso di assenza dell'argomento, deve mostrare i processi di tutti gli utenti (compresi quelli senza terminale di controllo) con anche le informazioni sul nome utente e ora di inizio**
  - **Se viene passato come argomento un intero (#lines) deve mostrare i primi #lines processi ordinati in senso crescente per ora di inizio del processo**

- **NOTA: non tutte le righe prodotte in output da** `ps` **hanno contenuto informativo rilevante**

21

Soluzione esercizio 1

```
if [ -z $1 ] #restituisce 1 se $1 è vuota
then
  ps u O+T
else
  ps u O+T | head -n `expr $1 + 1`
fi
```

22

cicli (`for`)

```
for <var> [in <list>]
   do
    <comandi>
   done
```

- list = lista di stringhe

23

esempi (`for`)

```
for i in *
# esegue per tutti i file nel direttorio corrente

for i                    # cioè: for i in $*
# esegue per tutti i parametri  di invocazione

#file crea
for i
   do > $i        # ridirezione di input su $i -> creazione
done
```

24

esempi (for)

```
for i in `cat f1` #per ogni parola di f1
   do <comandi >
done

for i in `ls s*`
  do cat $i | grep $2 | wc -w
done
```

25

cicli (`while`)

```
while <lista_comandi>
   do
    <comandi>
   done
```

- `while :` esegue un loop infinito:

```
while :
   do echo “e’ passato un minuto”
      sleep 60s
   done
```

esempio di file comandi

```
# nuovo comando: loop <directory>
# avvisa se sono stati creati o
# eliminati file in una directory

if [ $# -ne 1 ] # controllo formato
then
  echo “usage: $0 nomedir”
  exit 1 # formato errato
fi
```

27

esempio di file comandi

```
if [ ! -d $1 ] # controllo parametro
then
  echo “$1 is not a valid directory”
  exit 2 # parametro errato
fi

# creazione file temporaneo
if [ ! –e loop.tmp ]
then echo 1 > loop.tmp
fi
```

28

esempio di file comandi

```
while :
do
  sleep 5
  if [ `ls $1 | wc –w` -ne `cat loop.tmp` ]
  then
      ls $1 | wc –w > loop.tmp
      echo in $1 sono presenti `cat loop.tmp`
  file
  fi
done
```

29