Introduzione allo shell scripting

Massimo Camarda

II corso Linux

Perche' shell scripting?

- permette di automatizzare le operazioni di routine con facilita'
- studiare lo shell scripting equivale a studiare la shell (piu' o meno)
- non sempre si ha a disposizione un terminale grafico ed inoltre i comandi della shell sono comuni a tutte le distribuzioni linux e unix

Alcune parole sulla shell

- rappresenta “l'interfaccia” attraverso cui poter interagire col sistema (dispositivi hardware)
- personalizzabile attraverso il contenuto dei file di configurazione (solitamente /etc/profile oppure ~/.profile)
- si puo' suddividere in: “*shell interattiva*” (cioe' da riga di comando) e “*shell non-interattiva*” (argomento di questa lezione)

Personalizziamo la shell 1/2

- Utilizzo degli “*alias*”

  utilizzo: alias <nome>=<comando>

  es: alias cerca=grep

  alias cd_dir_name='cd /bin/...'

Personalizziamo la shell 2/2

- Variabili d'ambiente:
  sono variabili accessibili e visibili a qualunque sottoprocesso della shell
  utilizzo: export <varname>=[<value>]
  export

Ridirezione dei Canali 1/3

- Ogni programma sotto linux per convenzione comunica seguendo sempre un analogo schema di input/output, tale schema comprende tre canali:
  - standard input (stdin)
  - standard output (stdout)
  - standard error (stderr)
- Di default il primo e' associato alla tastiera gli ultimi due allo schermo, questo comportamento puo' pero' essere modificato.

Ridirezione dei Canali 2/3

- Ridirezione dell'input:
  sintassi generale: comando < filename
  in questo caso il comando ricevera' in ingresso l'output del file.
  - Es.
    - sort < prova.txt ordina prova.txt in modo crescente
    - grep “AA” < prova.txt selezione righe contenenti “AA”

Ridirezione dei Canali 3/3

- Ridirezione dell'output:
  sintassi generale: comando > nome_file
  in questo caso l'output invece di essere mostrato a schermo verra' scritto su file (molto utile in caso di output troppo grossi).
  - Es.

    ls > output.txt (un classico...)

    (sort < file_name.txt) > file_ordinato.txt

Script: introduzione

- cos'e' uno script?
  In generale uno script e' un file di testo contenente comandi per un interprete.
  utilizzo: <interprete> <script> <opzioni>
- In generale per semplicita' si preferisce indicare l'interprete direttamente nella prima riga della script.
  Nella prima riga: #!/bin/bash
  oppure: #!/bin/awk ecc.

Script: utilizzo 1/2

- Specificato l'interprete dei comandi all'interno allo script si puo' rendere lo scritp eseguibile per maggiore facilita' di utilizzo:
  > chmod +x <nomescript>
  Fatto questo sara' possibile richiamare lo script nel seguente modo:
  > ./<nomescript> <argomenti>

script: utilizzo 2/2

- Si puo' poi inserire la directory dove si trova lo script all'interno della variabile di ambiente “PATH”, in questo caso finale lo script sara' richiamabile da qualunque directory semplicemente digitando:
  ><nomescript> <arg>
  Per modificare “permanentemente” la variabile “PATH” (fare MOLTA attenzione)
  PATH=$PATH:<directory dello scritp>

Variabili

- In generale definire e richiamare una variabile e' molto semplice:
  <variabile> = [<valore>] specifica il valore
  ${variabile} ritorna il valore
- per assegnare l'output di un comando o di un'espressione aritmetica ad una variabile:
  <variabile>=$(comando)
  <variabile>=$((espressione numerica))

Parametri posizionali

- Quando si richiama uno script e' possibile passare dei parametri esterni, questi sono poi accessibili dall'interno dello script:

${n}	valore dell'ennesimo parametro
$*	insieme di tutti i parametri
$#	numero di parametri posizionali
$0	ritorna il nome dello script

Strutture per il controllo del Flusso

- Come in qualunque linguaggio di programmazione esistono strutture che permettono di alterare il normale flusso del programma:

if-then-else-fi
for-do-done
while-do-done
case select

IF <condizione>; then <oper> else <oper> FI

- Un utilizzo comune e' quello di definire la condizione come valore di ritorno di un comando:
  if <esecuz regolare comando>; then <>
  else <gestione dell'errore>
  fi
- E' possibile poi fare confronti multipli attraverso l'utilizzo dell' AND(&&) e OR(||), e della negazione della condizione con (!)

Utilizzo del comando “test” 1/2

- Un altro utilizzo molto comune e' quello di specificare la condizione attraverso l'utilizzo del comando “test”:
  test <condizione> oppure [<condizione>]
- Esempio:

```
#!/bin/bash
if test -e $1; then
echo “e' presente il file $1”
else echo “non e' presente il file $1”
```

Comando Test 2/2

- Alcune possibili opzioni:
  - -e <file> :<file> esiste
  - -s <file> :<file> esiste e non e' vuoto
  - -n <stringa> :<stringa> e' non nulla
  - !<condizione> :inverte il valore dell'opzione
  - -lt , -gt , -eq , -ne :confronti tra numeri e stringhe
- Le opzioni possibili per l'utilizzo di “test” sono tantissime.
  Una lista completa la si puo trovare al sito:
  http://cdrom.gnutemberg.org/main.html
  oppure semplicemente: man test

FOR <var> [in <list>] ; do <> done 1/2

- Il costrutto FOR e' differente da quello dei linguaggi tradizionali, nella sua forma base infatti non permette di specificare il numero di iterazioni. Ad ogni iterazione invece, viene attribuito ad <var> un valore in <list> e quindi si esegue il blocco <> (che tipicamente conterra' riferimenti a <var>), fino alla fine di <list>.

FOR <var> [in <list>] ; do <> done 2/2

- Esempio 1:

```
echo “questa e' la lista di file”
for nome_file in $(ls); do
    echo ${nome_file}
done
```

- Esempio 2:

```
for A in “1 2 3 4 5”; do
    echo $A
done
```

Funzioni(){} 1/2

- Anche in shell scripting, come in qualunque altro linguaggio conviene spezzare il codice in sottoprogrammi, piu' gestibili e riutilizzabili.
- Questo si ottiene attraverso l'uso di funzioni. In generale non e' necessario specificare nella dichiarazione i parametri di ingresso o di uscita, si utilizza invece il comando ${n} dall'interno della funzione, e il comando “export” per esportare le variabili.

Funzioni(){} 2/2

- Es:

```
#!/bin/bash
func_di_prova(){
    echo “il primo parametro e' ${1}”
    echo “il secondo e' ${2}”
    A=”definita dentro funzione”
    export A}

func_di_prova parametro1 parametro2
```