GLI SCRIPT DI SHELL

Shell Unix - Linux
comandi

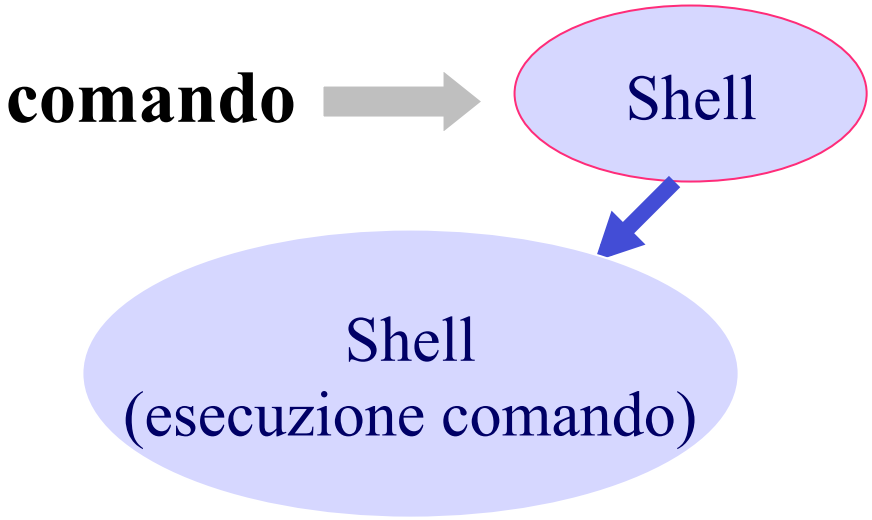
Interfaccia di alto livello tra utente e SO.

Il processore interpreta e mette in esecuzione comandi da: **1) standard input**

o 2) da file comandi

BASH= linguaggio comandi con elevato potere espressivo

- ogni comando richiede al nucleo l'esecuzione di una particolare azione
- i comandi esistono nel file system come files binari, generalmente eseguibili da tutti gli utenti **(/bin)**
- possibilità di realizzare nuovi comandi: ***programmazione in shell***



per ogni comando da eseguire la shell crea una shell figlia dedicata all'esecuzione del comando

Programmazione in shell: script …

Uno script di shell può essere editato con un qualunque editor di file (per esempio col **vi**).

```
Es:   vi stampa-home
              #!/bin/bash
               echo $HOME
```

Dopo aver editato lo script, dobbiamo salvarlo, uscire dall'editor ed infine eseguirlo. Il modo piu' semplice per eseguire lo script creato in BASH e' col comando **sh <nomefile>.**

```
ES:    mio-prompt> sh stampa-home
       /home/giuwal
```

Un altro modo per eseguire uno script dopo averlo editato, è quello di rendere eseguibile il file che lo contiene : `chmod +x <nomefile>`
poi si deve modificare il PATH in modo tale che comprenda la localizzazione del file, così da poterlo eseguire: `PATH=$PATH:$HOME`
ed infine, per lanciarne l'esecuzione, basta scrivere il nome dello script creato:

```
       mioprompt> <nomefile>
```

...riassumendo

Per realizzare uno script:

1 - Si scrive un file di testo con i comandi da far eseguire;

2 - Nella prima riga si mette #! e il percorso dell'interprete (di solito /bin/bash) (per comunicare la shell che eseguirà lo script);

3 - Si attiva (con *chmod*) il permesso e**X**ecute sul file che contiene lo script.

4. Si deve modificare il PATH.

…le variabili $ d'ambiente

Ogni processo in esecuzione ha un proprio ambiente definito in base a variabili di ambiente (identificate dal carattere $).

Le *variabili d'ambiente* sono un mezzo elementare e pratico di configurazione del sistema: i programmi, a seconda dei loro compiti e del loro contesto, leggono alcune variabili e variano il loro comportamento in base al contenuto di esse.

Esempio: **la variabile PATH**

La variabile PATH indica tutte le directory dove la shell deve cercare i file eseguibili . E' una variabile a tutti gli effetti:

```
$ echo $PATH
/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr
/X11R6/bin:/usr/games:/home/giangy/bin:/usr/sbin
:/usr/X11R6/bin:/usr/games
```

Variabili d'ambiente

L'interprete prevede delle variabili, come la maggior parte dei linguaggi di programmazione, che in gergo si definiscono anche parametri.

$# numero di argomenti passati al comando

$* o $@ tutti gli argomenti passati

$- opzioni fornite all'interprete

$? valore restituito dall'ultimo comando eseguito

$$ identificativo di elaborazione dell'interprete

$! Identificativo di elab. dell'ultimo comando iniziato con &

$HOME argomento di default per cd

$IFS elenco dei caratteri separatori di parole negli argomenti

$MAIL file che, quando viene modificato, invia il messaggio "You have mail"

$PATH elenco delle directory in cui cercare i comandi

$PS1 stringa del simbolo che segnala il prompt (per default il segno è $)

$PS2 stringa del simbolo che segnala che la riga di comando continua; per default il segno '>'

…le variabili

Il nome e il valore di alcune variabili sono predefinite dalla shell, e non possono essere modificate direttamente, altre sono definite come nome, ma non assegnate (sono utilizzate implicitamente da alcuni comandi):

- *Predefinite ed assegnate*

PPID parent del processo
PWD current working directory
UID user ID
EUID effective user ID
……..

- *Predefinite, non assegnate*

PATH pathname fra cui cercare i comandi
HOME home directory del current user
PS1 primary prompt string
…….

Gestione delle variabili in BASH

Variabili di shell

Una variabile è definita la prima volta che viene usata, anche se non contiene valori. L'assegnazione di un valore si ottiene con una dichiarazione quale: **`nome_di_variabile=[valore]`**

Esempio:

```
[giangy@homer giangy]$ pluto=cane_di_topolino
[giangy@homer giangy]$ echo $pluto
                       cane_di_topolino
```

Le variabili di shell sono visibili limitatamente all'ambito della shell stessa. Quindi, i comandi interni alla shell stessa sono al corrente delle variazioni sulle variabili mentre i programmi che vengono avviati non ne risentono.
Per i programmi esterni le variabili devono essere **esportate**.
L'esportazione delle variabili si ottiene con il comando interno ***export.***

Esempi

```
$ PIPPO="ciao"
$ export PIPPO
Oppure : $ export PIPPO="ciao"
```

SET

Il comando ***SET*** da solo visualizza il contenuto di tutte le variabili d'ambiente e anche di quelle definite dagli utenti.

Il comando **`set $(`*`comando`*`)`** memorizza l'output del comando tra apici in variabili ($1 , $2, ecc.) che si chiamano ***parametri posizionali***.

ESEMPIO:

$ **date**

Sat Oct 1 06:05:18 EDT 1983

$ **set $(date)** (cioè memorizza l'output di date nei parametri posizionali)

$**echo $1** (cioè stampa il primo campo di date)

Sat

$**echo $6** (cioè stampa il sesto campo di date)

1983

Il comando **`UNSET <nomevar>`** permette di cancellare le variabili create dall'utente.

Caratteri di protezione

A volte magari abbiamo bisogno di scrivere effettivamente caratteri come: **`$ * { [ . . . }`**
Esistono 3 modalita':
1. Escape \ 2. Apici singoli 3. Apici doppi

La `back-slash(\)` rappresenta il carattere di *escape*. Serve per preservare il significato letterale del carattere successivo, cioè evitare che venga interpretato diversamente da quello che è veramente.

`Esempio:  ls -l \$fff`

cerca di leggere il file che si chiama proprio `$fff`

Usato da solo (quindi con uno spazio dopo) serve per continuare una linea (nonostante si vada a capo)

```
Ad Esempio: $ ls -l \
            supercalifragilistichespiralidoso
```

viene interpretato come un unico comando su un' unica riga

Apici Singoli

Racchiudendo dei caratteri tra apici semplici (’) si mantiene il valore letterale di questi caratteri.
Un apice singolo non può essere contenuto in una stringa del genere.

Esempio

```
$ echo ’$0 $1 \ ... restano "inalterati".’
      $0 $1 \ ... restano "inalterati".
```

Apici Doppi

Racchiudendo una serie di caratteri tra una coppia di apici doppi si mantiene il valore letterale di questi caratteri, a eccezione di $, ‘ e \.

Esempio

```
$ echo "Il parametro ‘$’0 contiene: \"$0\""
Il parametro $0 contiene: "-bash"
```

Costrutti di Controllo (1)

Per quanto riguarda i costrutti di controllo questi sono diversi a seconda che si tratti di ***Bourne shell o C shell.***
La tabella che segue ne da la lista mettendo in evidenza analogie e differenze.

COSTRUTTO	BOURNE SHELL	C-LIKE SHELL
Ciclo For	**for** name **in** word **do** .... **done**	**foreach** var (wordlist) .... **end**
Alternativa-1	**if** command; **then** .... ;	**if** (expr) command

Alternativa-2	**if** command; **then**
	.... ;
	elif command;
	then
	.... ;
	else
	.... ;
	fi
		if (expr) **then**
		....
		else if (expr2)
		then
		....
		else
		....
		endif

Costrutti di Controllo (2)

COSTRUTTO	BOURNE SHELL	C-LIKE SHELL
Assegnazione	var=value var[n]=word	**set** var = value **set** var[n] = word
Selezione	**case** word **in** pattern) .... ;; pattern) .... ;; .... **esac**	**switch** (string) **case** label: .... **breaksw** **endsw**
Ciclo While	**while** list **do** .... **done**	**while** (expr) .... **end**

BASH

Script che esegue il comando *`ls -l <nome_file>`* dove <nome_file> puo' essere un insieme (ad esempio *.c), listando anche il contenuto di eventuali directory.

```
#! /bin/bash
#script dir
if [ "$#" = 0 ] ; then          # numero dei parametri in decimale
echo nessun file else
for i       # essendo omesso il resto del comando (in <set>),
            # $i assume uno alla volta il valore dei parametri posizionali
do ls -l $i; done
fi
```

Dare il comando dir *.c. La shell espande in un buffer, prima di eseguire lo script dir, la stringa "*.c", ottenendo un insieme anche vuoto di file. Quindi esegue lo script dir.

Esempio di script bash

Costruire uno shell script che accetti come argomento una delle seguenti lettere: D, W, L, P e restituisca rispettivamente: la data di oggi, l'elenco degli utenti collegati, la lista dei file e la directory corrente. Controllare anche che l'utente inserisca un argomento e, in caso negativo, suggerisca le scelte possibili.

```
#scelta dei comandi
#! /bin/bash
echo “D - data corrente;”
echo” W - utenti collegati;”
echo” L - lista dei file; “
echo “P - directory corrente; “
echo “Inserire la scelta: “
case $1 in
        'D') date;;
        'W') who;;
        'L') ls -l | more;;
        'P') pwd;;
        *) echo "Mettere le scelte giuste: D,W,L o P";;
esac
```

ESEMPIO

Il programma stampa a video in successione il nome dei parametri presenti sulla linea di comando, insieme al numero che ne indica la posizione

```
#!/bin/bash
count = 1
while [ “$*” != “ ” ]; do
    echo “Il parametro $1 ha il numero di ordine $count”
    shift
    appoggio1 = $count
    appoggio2 = 1
    count = $[$appoggio1+$appoggio2];
done
```

…READ

Per leggere l'input dell'utente si usa il comando è **READ** che legge una riga di testo dal file standard e la assegna (senza l'invio finale) come valore della variabile indicata:

```
$ read greeting
CIAO   (bisogna digitare qualcosa)
$ echo $greeting
CIAO
```

Se non viene specificata la variabile, il comando crea la variabile d'ambiente REPLY

…IL COMANDO TEST

Il comando **test** e' molto usato, specialmente nella sua forma abbreviata **[expr]**.
Si possono ad esempio verificare proprieta' dei file, o uguaglianza di stringhe (non valori numerici).

Alcuni esempi:

`test -d file`	vero se file esiste ed e' una directory.
`test -f file`	vero se il file esiste ed e' un file di dati.
`test -e file`	vero se il file esiste.
`test -L file`	vero se il file esiste es e' link simbolico.
`test -r file`	vero se il file esiste ed e' leggibile.
`test -x file`	vero se il file esiste ed e' eseguibile.
`test stringa1 = stringa2`	vero se uguali.
`test stringa1 != stringa2`	vero se diversi

BASH

Il comando *test expr* puo' anche essere scritto come: *[expre]*

ESEMPIO:

```
#! /bin/bash
# script di login
# attenzione ai blank nella prossima linea
if [ "$LOGNAME" = "root" ];  then
      echo " Welcome dear $LOGNAME"
      if [ -f $HOME/hello ]; then
                  echo $HOME/hello
      fi
fi
```

ESEMPIO:

```
#! /bin/sh
# script sbh
if [ "$SHELL" = "/bin/bash" ]; then
       PS1="sono la shell bash > "
fi
# fine
```

BASH

ESEMPIO:

```
#! /bin/sh
echo -n "stringa ? " # non va a capo
read VAR
case $VAR in
       "ALPHA" | "BETA" | "GAMMA" )
            echo maiuscolo ;;
       "alpha" | "beta" | "gamma" )
            echo minuscolo ;;
      * )
            echo altro ;;
esac
```

ESEMPIO

```
#! /bin/bash
# Script che ogni 30 secondi verifica se un file
# esiste.
# provare a scrivere uno script che vive sempre e
# notifica
# se un file esiste (solo la prima volta) o non esiste
# e' stato cancellato
# script s2
until [ -f $1 ]; do sleep 30;
Done
# oppure
until [ test -f $1 ]; do sleep 30;
done
echo "Il file $1 esiste"
```