



COMANDI DI CONSOLE BASH POCO CONOSCIUTI

SET

Il comando "set" permette di modificare molti comportamenti e funzionalità della bash. Mettere il carattere "-" davanti all'opzione per abilitare una funzionalità. Per disabilitarla utilizzare il carattere "+". Lanciato senza parametri il programma restituisce le impostazioni generali della shell, comprensive di variabili e funzioni che si riferiscono all'ambiente. Per fare in modo che le variabili create vengano automaticamente esportate è possibile attivare l'opzione "a":

```
$ set -a  
$ VARIABLE="VALORE"  
$ bash  
$ echo $VARIABLE  
VALORE
```

Per fare in modo che i comandi lanciati all'interno del sistema siano stampati, sarà sufficiente attivare l'opzione "x", prima dell'output di ogni comando successivo apparirà il comando effettivamente lanciato.

```
rasca@anomalia:~$ set -x  
++ echo -ne '\033]0;rasca@anomalia: ~\007'  
rasca@anomalia:~$ ls  
+ ls --color=auto  
...  
...
```

il comando lanciato è quindi in realtà un alias dello stesso con l'opzione "--color=auto". Tutte le altre funzionalità relative al comando set si trovano nella voce omonima della "man page" del comando bash.

DECLARE

Il comando declare è utilizzato per la dichiarazione delle variabili. Generalmente in bash una variabile viene dichiarata automaticamente con la sua definizione:

```
$ FOO=bar
```



dopodiché è utilizzabile con il nome \$FOO. Il comando declare consente in fase di dichiarazione di specificare il tipo di variabile oltre che alcuni attributi inerenti al suo utilizzo. Ad esempio, per dichiarare un array utilizzando il comando declare si utilizzerà l'opzione "-a":

```
$ declare -a elementi=("primo" "secondo" "terzo")
```

mentre l'opzione "-i" dichiarerà una variabile di tipo integer:

```
$ declare -i numero=10
```

declare torna utile in fase di definizione anche per esportare automaticamente le variabili, attraverso l'opzione "-x":

```
$ declare -x -i numero=10
```

questo comando ottiene lo stesso effetto di:

```
$ export numero=10
```

ma in più aggiunge anche la definizione del tipo. Infine, attraverso l'opzione "-r" è possibile rendere la variabile readonly, ossia non scrivibile all'interno della sessione.

TYPE

Il comando type può essere utilizzato per conoscere le caratteristiche relative agli argomenti passati. Attraverso l'opzione "-t" viene restituito il tipo di parametro passato:

```
$ type -t ls  
alias  
$ type -t /bin/ls  
file  
$ type -t echo  
builtin  
$ type -t set_prefix  
function  
$ type -t while  
keyword
```

E' da notare la differenza tra "ls" e "/bin/ls", rispettivamente un alias (relativo ad un file eseguibile) ed un file eseguibile. Ad un comando di tipo builtin non corrisponde necessariamente un file



eseguibile, mentre le ultime due righe rappresentano rispettivamente una funzione disponibile nell'ambiente ed una parola chiave relativa al linguaggio bash.

Un'altra opzione supportata dal comando `type` è "-a" che indica tutte le corrispondenze al parametro passato. Ad esempio, relativamente ai comandi precedentemente esposti i risultati saranno i seguenti:

```
$ type -a echo  
echo is a shell builtin  
echo is /bin/echo  
$ type -a ls  
ls is aliased to `ls --color=auto`  
ls is /bin/ls  
$ type -a set_prefix  
set_prefix is a function  
set_prefix ()  
{  
  [ -z ${prefix:-} ] || prefix=${cur%/*}/;  
  [ -r ${prefix:-}CVS/Entries ] || prefix=""  
}  
$ type -a while  
while is a shell keyword
```

il comando `echo` viene indicato come interno e disponibile inoltre con l'eseguibile `/bin/echo`, lo stesso avviene per il comando `ls` per il quale viene indicata la definizione dell'alias oltre che il path del file effettivo. Infine la funzione `set_prefix` viene stampata nella sua interezza, mentre per `while` rimane l'indicazione di keyword.

EXEC

Il comando `exec` permette di lanciare un comando utilizzando il process ID (PID) della shell corrente.

```
$ exec xterm
```

alla chiusura della shell del programma `xterm`, anche la console verrà automaticamente chiusa.

JOBS

Se il comando interno `bg` viene utilizzato per mandare un processo in background (cioè sullo sfondo del sistema e non in primo piano, mantenendone l'esecuzione) ed il comando interno `fg` viene utilizzato invece per mandare in foreground (quindi, in primo piano) un processo, il comando `jobs` permette di effettuare il listato dei processi e di conoscere informazioni relative a questi.

Supponendo di avere due script denominati `script` e `script1` lanciati entrambi automaticamente in background attraverso il carattere speciale `&` alla fine dell'invocazione:



```
$ ./script &  
[1] 28548  
$ ./script1 &  
[2] 29261
```

Attraverso "jobs" potremo conoscere lo stato dei processi in sospenso:

```
$ jobs  
[1]- Running          ./script &  
[2]+ Running          ./script1 &
```

con l'opzione "-l" si potrà conoscere sia identificativo che PID:

```
$ jobs -l  
[1]- 28548 Running          ./script &  
[2]+ 29261 Running          ./script1 &
```

Oppure solamente il PID:

```
$ jobs -p  
28548  
29261
```

Su questi processi sarà possibile effettuare operazioni di kill:

```
$ kill 28548  
$  
[1]- Terminated          ./script
```

Oppure operazioni di foregrounding/backgrounding, attraverso i già citati comandi interni fg e bg, considerando invece l'identificativo:

```
$ fg 2  
./script1
```

Il tutto grazie ai dati resi disponibili dal comando interno jobs.

I comandi interni alla bash illustrati e molti altri sono tutti ottimamente documentati all'interno della già citata man page della Bash (`$ man bash`), più propriamente nel capitolo "COMANDI INCORPORATI DELLA SHELL". Nella seconda parte di questo articolo verranno trattate utility di sistema come paste, tee, apropos e molte altre, generalmente poco utilizzate, ma di indubbio interesse.

Fonte: <http://www.tuxjournal.net/?p=9788>

Articolo originale scritto da: **Raoul Scarazzini** – MiaMammaUsaLinux.org