Come funziona Linux: introduzione alla shell

Finalmente siamo arrivati ad analizzare la shell, l'interprete di comandi che funge da interfaccia fra l'utente e il sistema operativo. Sebbene esistano per Linux diverse interfacce grafiche, anche evolute, l'approccio testuale offre delle possibilità ancora irraggiungibili mediante il solo uso del mouse.

Quarta parte

di Giuseppe Zanetti

La shell

Attraverso la shell l'utente può inserire comandi, avviare programmi e comunicare col sistema operativo. Una shell che si rispetti mette inoltre a disposizione un ambiente di lavoro in cui l'utente può definire delle variabili che influenzano il funzionamento degli altri programmi ed un linguaggio di programmazione, che può essere utilizzato inserendo i comandi desiderati direttamente sulla "linea di comando" oppure per creare dei piccoli programmi (shell script). Non esiste un'unica versione della shell, ma può essere utilizzato come shell qualunque programma in grado di interagire con l'utente. E' per esempio possibile fare in modo che quando l'utente si collega al sistema ed inizia una sessione di lavoro, al posto della shell standard di Linux venga lanciato un menu di scelte simile a quello utilizzato nelle BBS oppure un sistema grafico. Limitandosi a considerare le classiche shell a linea di comando, ne esistono decine di versioni diverse: gli affezionati del linguaggio C possono ad esempio utilizzare la C SHell (csh), che offre un linguaggio di programmazione simile a quello di Kernigan e Ritchie, mentre i patiti della sicurezza possono installare Restricted SHell (rsh), che limita l'utente all'interno della propria directory e permette di eseguire solamente un insieme limitato di comandi.

In questo articolo faremo riferimento alla bash (Bourne Again SHell), in quanto essa, oltre ad essere diventata lo standard *de facto* in Linux, estende ed è compatibile verso il basso con la Bourne Shell presente in tutti i sistemi UNIX. I lettori interessati ad estendere gli argomenti trattati potranno perciò utilizzare con profitto le decine di testi e documenti disponibili per quest'ultima. Ovviamente il documento principe a cui fare riferimento è il manuale della bash (man bash), eventualmente nella versione tradotta in italiano.

La shell bash mette a disposizione alcune funzioni principali che analizzeremo nel seguito dell'articolo, fra cui:

- ✓ linea di comando
- ✓ ambiente (environment)
- ✓ sostituzione dei nomi
- ✓ ridirezione dell'input/output.

Scelta della shell di login

Quando un utente esegue il login sulla macchina, viene lanciato il programma indicato nell'ultimo campo di /etc/passwd.

```
beppe:AQ23HADsIQsvo:306:100:Giuseppe
Zanetti:/home/beppe:/bin/bash
```

Nel caso esso sia vuoto, viene eseguita la shell standard /bin/sh (che in quasi tutte le distribuzioni di Linux è un link simbolico a /bin/bash).

Ogni utente può scegliere quale programma utilizzare come shell di login, utilizzando il comando chsh.

```
# chsh beppe
Changing shell for beppe.
New shell [/bin/sh]: /bin/bash
Shell changed.
```

La possibilità di usare come shell qualunque programma permette di creare degli utenti che compiano in modo automatico certe operazioni, ad esempio lo shutdown della macchina:

```
shutdown:x:6:0:Utente Shutdown:/sbin:
/sbin/shutdown
```

Per motivi di sicurezza, è possibile utilizzare come shell solamente i programmi il cui percorso è contenuto nel file /etc/shells.

Interazione dell'utente con bash

La bash, rispetto alla Bourne Shell, mette a disposizione alcune funzioni interessanti che facilitano e velocizzano moltissimo l'interazione da parte dell'utente. Una volta che si sia acquisita una certa pratica, compiere operazioni con la shell (spostamento di file,

...) risulta molto più semplice e veloce che utilizzando una qualunque interfaccia grafica.

L'history dei comandi

La prima cosa comoda che si incontra in bash è sicuramente la possibilità di spostarsi con i tasti cursore a destra e a sinistra per modificare una linea di comando che si sta inserendo. Nella vecchia Bourne Shell invece l'unica operazione possibile era quella di cancellare l'ultimo carattere inserito. E' poi possibile, utilizzando i cursori verso l'alto e verso il basso, navigare nell'history (storia) dei comandi inseriti precedentemente. In questo modo si può ripetere un comando oppure modificarlo. L'history viene salvata nel file .bash_history, presente nella home directory di ogni utente che utilizza come shell la bash. Tale nome è modificabile assegnando un opportuno valore alla variabile HISTFILE. Se non viene limitato in dimensione - ad esempio mediante la variabile HISTFILESIZE oppure cancellandolo mediante un opportuno comando nello script .bash_logout che viene eseguito al termine della sessione di lavoro - il file che contiene l'history ha la tendenza a crescere abbastanza velocemente.

Questa situazione porta ad un rallentamento nella partenza della sessione di lavoro, in quanto il file .bash_history viene caricato in memoria. Può essere utile cancellare il file di history anche per motivi di privacy, in quanto esso tiene traccia di tutte le operazioni da noi compiute in precedenza.

Oltre che con i tasti cursore, è possibile richiamare un comando precedente anche utilizzando il carattere !, seguito dalle prime lettere del comando che si desidera ripetere. Ad esempio per ripetere l'ultimo comando cp è sufficiente scrivere:

```
$ !cp
cp /home/beppe/doc/beppe/poets/il_gio-
co. txt /tmp/
$
```

E' anche possibile recuperare ad esempio la 24-esima linea di comando presente nell'history, utilizzando la scrittura !24:

```
$ !24
ls -la
```

Per conoscere il contenuto dell'history si utilizzi il comando "history", eventualmente bloccando l'output ad ogni schermata mediante more:

```
$ history|more
   21  vi pippo.c
   22  cp pippo.c pluto.c
   23  vi pluto.c
   24  make
```

Ulteriori informazioni sull'utilizzo dell'history possono essere, al solito, reperite nel manuale in linea della shell.

Gli alias

Il meccanismo dell'history semplifica moltissimo la vita dell'utente, ma se usato troppo in fretta e con poca attenzione può essere causa d'errori anche gravi. Ad esempio, ripetendo per errore un comando di copia dopo aver modificato il file, tutte le modifiche effettuate nel frattempo vengono perdute, in quanto si sovrascrive il file con una vecchia versione. I guai più seri derivano comunque dall'utilizzo dell'history associata al comando rm. Per questo motivo non è una brutta idea definire nello script .profile - che viene eseguito all'inizio della sessione di lavoro (ne riparleremo in seguito) - i comandi rm e cp come degli alias verso "cp -i" e "rm -i". L'opzione -i fa in modo che venga richiesta conferma prima di eseguire la modifica del file.

Per creare un alias è sufficiente inserire nel file .profile i seguenti comandi:

```
alias rm="rm -i"
alias cp="cp -i"
alias mv="mv -i"
```

Per avere la lista degli alias definiti si utilizzi il comando alias senza parametri.

```
$ alias
alias cp='cp -i'
alias dir='ls -la'
```

Completamento dei nomi

Un'altra funzione molto interessante della bash è la possibilità di completare i nomi utilizzando il tasto TAB. E' sufficiente scrivere una parte del nome del comando che lo identifichi univocamente e premere TAB affinché il nome venga completato automaticamente dalla shell. Nel caso i caratteri inseriti non siano sufficienti ad identificare un singolo comando in modo univoco, verrà emesso un suono, che invita ad inserire ulteriori caratteri. Premendo ulteriormente TAB verranno mostrati tutti i comandi che iniziano con i caratteri inseriti, in modo da facilitare la scelta all'utente.

Ad esempio, dovendo inserire il comando mkdirhier, sarà sufficiente scrivere dir e premere il tasto tab per completare il nome. Non essendo le lettere inserite sufficienti ad identificare univocamente il comando richiesto, sarà necessario inserire anche ulteriori lettere.

```
$ mkd<TAB>
mkdexe mkdict mkdir mkdirhier mkdosfs
$ mkdir<TAB>
mkdir mkdirhier
$ mkdirh<TAB>
$ mkdirhier
```

Un discorso analogo vale per i nomi dei file. In questo caso il meccanismo risulta ancora più efficace, in quanto permette di completare velocemente nomi molto lunghi ed evita moltissime battute sulla tastiera. L'esempio appena fatto non è dei migliori, ma si pensi di dover inserire a mano il comando "multiload_applet". Utilizzando il meccanismo descritto sarà sufficiente scrivere "mul<TAB>".

Espansione della tilde

Una ulteriore funzione interessante è l'espansione di un nome preceduto dal simbolo della tilde nel percorso relativo alla home directory dell'utente corrispondente:

```
$ ls ~beppe/doc (equivale a ls /disk2/
utenti/home/beppe/doc)
divinacommedia.doc decameron.txt
odissea.rtf
```

Personalizzare la shell

Quando l'utente inizia la propria sessione di lavoro la shell esegue il file /etc/profile, che deve essere uno script di shell ed avere gli opportuni permessi di eseguibilità. L'amministratore di sistema può predisporre questo file in modo da creare alias validi per tutti gli utenti del sistema, oppure per verificare e segnalare la presenza di mail o la quota di disco residua, per definire eventuali variabili d'ambiente, ecc...

Anche ogni singolo utente ha la possibilità di personalizzare la partenza della shell, mediante il file .profile contenuto nella propria home directory (es:

/home/beppe/.profile). Come /etc/profile, anche i .profile personali devono essere degli shell script ed avere gli opportuni permessi di eseguibilità da parte dell'utente. Bash utilizza con scopo simile anche i file .bashrc, .bash_profile e .bash_login (per le differenze si faccia riferimento al manuale). Esiste infine uno script, .bash_logout, che viene eseguito al termine della sessione di lavoro.

La linea di comando

La shell si presenta all'utente attraverso un prompt ("sono pronto"), che per un utente normale è di solito il simbolo "$", mentre per root è "#" (vedremo in seguito che è possibile personalizzare il prompt utilizzando le variabili d'ambiente PS1 e PS2). Ad esso si può rispondere inserendo una "linea di comando", composta da uno o più comandi, separati dal simbolo ";", i quali vengono passati al sistema per essere eseguiti.

Ad esempio la linea che segue lancia i comandi "cd /bin" e "ls" in sequenza:

```
cd /bin ; ls
```

Alcuni comandi fondamentali sono integrati (builtin) nella shell stessa, mentre gli altri vengono cercati in una lista directory contenuta nella variabile d'ambiente PATH (es: /bin:/usr/bin:/usr/local/bin). E` inoltre possibile inserire in una linea di comando il percorso completo per raggiungere un dato programma.

Possono essere utilizzati comandi appartenenti ad una delle seguenti classi:

- comandi interni alla shell
- programmi compilati
- shell script realizzati nel linguaggio interno alla shell
- script eseguiti mediante un interprete.

Ovviamente sia i programmi che gli script, per poter essere eseguiti, devono avere i giusti permessi (es.: chmod u+x mioprogramma).

Quando si digita una parola sulla linea di comando, la shell è in grado di riconoscere automaticamente se si tratta di un proprio comando interno. In caso contrario essa verifica se in una delle directory indicate dalla variabile PATH è contenuto un file col nome richiesto che abbia gli opportuni permessi di esecuzione. Se tale file esiste, il sistema operativo riconosce se si tratta di un programma in formato eseguibile conosciuto (a.out, ELF, classe java, ...) oppure di uno script. Uno script ben scritto deve contenere nella prima riga l'indicazione del programma da utilizzare come interprete per farlo girare, ad esempio uno script nel linguaggio della Bourne Shell dovrebbe avere come prima riga la seguente sequenza di caratteri:

```
#!/bin/sh
```

Uno script in perl invece dovrebbe iniziare con una scrittura che indichi al sistema operativo il percorso dove è installato l'interprete di questo linguaggio:

```
#!/usr/bin/perl
```

E così per ogni altro possibile linguaggio interpretato.

Caratteri con funzioni particolari

Nella linea di comando possono essere utilizzati dei caratteri con funzioni speciali. Abbiamo già visto nelle scorse puntate come ad esempio il simbolo "&" permetta di eseguire un programma in background oppure ">" ridirezioni l'output di un comando in un file.

La sintassi di un tipico comando di Linux è la seguente:

```
comando -selettore1 -selettore2
argomento1 argomento2 ...
```

Il simbolo "-" non viene gestito direttamente dalla shell, ma è una convenzione che serve ad indicare al comando che la lettera che segue deve essere trattata come un selettore (switch, modificatore) di opzione. Ad esempio il selettore -l nella linea di comando "ls -l" indica al comando ls che la lista dei file deve essere visualizzata nel formato lungo. E` consuetudine scrivere i programmi in modo che più selettori possano essere raggruppati sotto un unico simbolo "-", in modo che ad esempio la scrittura "ls -la /bin" sia equivalente a "ls -l -a /bin". L'ordine dei selettori di solito (esistono dei programmi che fanno eccezione) non è importante, cosicché è indifferente scrivere "ls -l -a" oppure "ls -a -l" (o anche "ls -la" o "ls -al"). Per evitare errori di interpretazione fra selettori di opzione ed argomenti, è buona norma non utilizzare il simbolo - come primo carattere nel nome di un file. Se si tenta ad esempio di cancellare il file "-pippo" mediante il comando "rm -pippo", si ottiene un messaggio di errore, in quanto tale linea di comando viene interpretata come se si volessero utilizzare nel comando rm le opzioni -p -i -p -p -o. Per ovviare a questo inconveniente si può utilizzare un trucco simile a quello che abbiamo visto per cancellare un file nascosto, ovvero indicare il percorso completo del file, eventualmente in modo relativo alla directory corrente "rm ./-pippo". In alternativa è possibile utilizzare il selettore "—", che indica al comando di interpretare tutti gli argomenti che seguono come parametri invece che come selettori (es: "rm — -pippo").

Attenzione, perché, essendo tale funzione interpretata a livello di ogni singolo programma, non è detto che essa funzioni sempre.

I comandi scritti nell'ambito del progetto GNU offrono la possibilità di indicare alcuni selettori anche in modo esteso, ad esempio "ls —format=long —all /bin" invece di "ls -la".

Il quoting dei caratteri speciali

Volendo utilizzare in una linea di comando eventuali caratteri speciali con il loro valore letterale, è necessario "quotarli", ad esempio racchiudendoli fra virgolette:

```
$ vi "food&drink.txt"
```

Ciò vale soprattutto per eventuali spazi, che altrimenti verrebbero interpretati come separatori fra i diversi parametri. Se si desidera copiare il file "ciao mondo.txt", il comando seguente non è corretto, in quanto i parametri che seguono il nome del comando cp vengono interpretati come "ciao" e "mondo.txt"

```
$ cp ciao mondo.txt /tmp
```

Un metodo corretto di scrivere il comando è il seguente:

```
$ cp "ciao mondo.txt" /tmp
```

In alternativa alle virgolette è possibile utilizzare il singolo apice ' (da non

confondere con la versione rovesciata ` che per la shell ha un significato diverso). Il singolo apice prende il nome di "quoting forte" (rispetto alle virgolette che sono il "quoting debole"), in quanto considera anche i caratteri speciali contenuti al proprio interno con il loro valore letterale. Una conseguenza di ciò è che non vengono espanse eventuali variabili:

```
# echo '$TERM'
$TERM

# echo "$TERM"
xterm
```

Un altro metodo di quoting consiste nel far precedere il carattere speciale con il simbolo di escape \:

```
$ echo drink\&food.txt
drink&food

$ echo \*\*\* \"ciao\" \*\*\*
*** "ciao" ***

$ cp ciao\ mondo.txt /tmp
```

Lo stesso carattere di "a capo" può essere quotato mediante uno dei meccanismi descritti:

```
# echo "ciao
> mondo"
ciao
mondo
```

Tale possibilità viene spesso utilizzata negli shell script per spezzare una linea lunga in più parti al fine di migliorarne la leggibilità:

```
# echo io sono un \
> esempio di linea \
> lunga ; date
io sono un esempio di linea lunga
Thu Mar  9 17:45:28 CET 2000
```

Il seguente esempio mostra un errore frequente:

```
echo Ciao Giuseppe, come stai ?
```

Il comando echo stampa in output tutti i parametri che gli sono stati passati in ingresso, perciò ci si aspetterebbe che in questo caso l'output fosse il seguente:

```
Ciao Giuseppe, come stai ?
```

In realtà il simbolo ? è per la shell una "espressione regolare" (verranno spiegate nel prossimo paragrafo) e viene sostituita da tutti i nomi dei file formati da un solo carattere. In molti casi non si noterà l'errore, in quanto nella directory corrente non esistono file con tale nome.

Se nella directory corrente esistessero ad esempio i file "a", "b" e "X" l'output generato dal comando sarebbe però:

```
Ciao Giuseppe, come stai X a b
```

Una forma più corretta di scrivere la linea di comando precedente consiste perciò nel racchiudere il tutto fra virgolette, per evitare che vengano interpretati eventuali caratteri speciali:

```
echo "Ciao Giuseppe, come stai ?"
```

Una cosa da notare è che, mentre prima venivano passati al comando echo più parametri ("Ciao", "Giuseppe,", "come", "stai", più il risultato dell'espansione della espressione regolare "?"), in questo caso viene passato l'unico parametro "Ciao Giuseppe, come stai ?". Il fatto che l'output sia simile è dovuto al funzionamento di echo, che separa l'output mediante spazi.

Le espressioni regolari

Le espressioni regolari permettono di abbreviare l'introduzione di più nomi di file utilizzando delle espressioni in sostituzione di gruppi di caratteri. L'espressione *txt indica ad esempio in modo sintetico tutti i nomi di file che terminano con txt (l'espressione "DOS" *.txt indica invece i file che terminano con ".txt"; ci si ricordi che in UNIX il carattere "." non ha un significato particolare).

Le espressioni regolari prendono il nome dal fatto che mediante esse è possibile individuare tutte le stringhe di una particolare classe di linguaggi formali detti "linguaggi regolari".

La sostituzione dell'espressione con i file avviene filtrando tutti e soli i nomi di file che soddisfano l'espressione richiesta (**Figura 1**). Tale operazione prende il nome di "espansione delle espressioni regolari" o "globbing".

Una espressione regolare è formata da caratteri che mantengono il loro valore letterale (ad esempio le lettere dell'alfabeto inglese) e da caratteri particolari, detti anche caratteri jolly o wildcard (la "matta" nel gioco delle carte può essere utilizzata per sostituire ogni altra carta).

Pur essendo quello di espressione regolare un concetto abbastanza comune nell'informatica, purtroppo nella pratica non vi è uniformità di implementazione ed il modo di scrivere una espressione non è sempre perfettamente lo stesso per la shell, il comando grep o i linguaggi perl o SQL.

Nel caso della shell, i principali caratteri speciali che possono essere utilizzati per comporre espressioni regolari sono i seguenti:

- `*` indica un qualunque gruppo di caratteri, compresa la stringa nulla ed escluso il carattere . all'inizio del nome di un file nascosto,
- `?` sostituisce un carattere qualunque non nullo, escluso il punto iniziale nel nome di un file nascosto,
- `[...]` sostituisce qualunque dei caratteri racchiusi dentro alla parentesi (es: [abc]). E' possibile specificare degli intervalli di caratteri, ad esempio [a-z] per indicare tutte le lettere minuscole, oppure [A-Z] per indicare tutte le lettere maiuscole. I caratteri - e] vengono considerati col loro valore letterale solamente se compaiono come primo o ultimo carattere,
- `[!...]` sostituisce qualunque carattere, esclusi quelli racchiusi dentro la parentesi. Al posto di ! può essere anche utilizzato il carattere ^.

Vediamo alcuni esempi di espressioni unitamente ad una breve de-

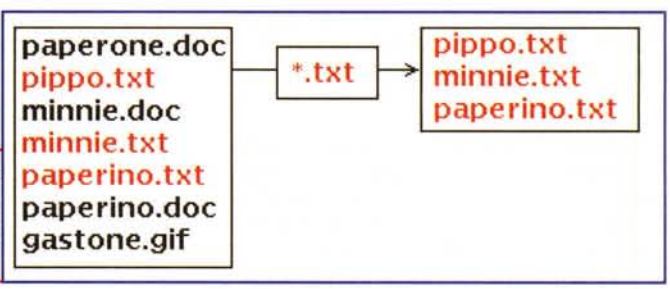

Figura 1 - L'espansione di una espressione regolare avviene filtrando i nomi dei file che la soddisfano.

scrizione del loro significato ed ai nomi di file che le soddisfano:

*	tutti i file	gastone.gif minnie.txt paperino.txt pippo.txt minnie.doc paperino.doc paperone.doc
pippo.txt	identifica il solo file pippo.txt	"pippo.txt"
.txt	file che terminano con "txt"	minnie.txt paperino.txt pippo.txt
pa*	file che iniziano con "pa"	paperino.doc paperino.txt paperone.doc
p*txt	file che iniziano con "p" e terminano con "txt"	paperino.txt pippo.txt
?????????	file col nome di 9 caratteri	pippo.txt
????????.doc	file col nome di 12 caratteri che terminano con ".doc"	paperino.txt paperone.txt
[mg]*	file che iniziano con "m" oppure "g"	gastone.gif minnie.doc minnie.txt
[!mg]*	file che non iniziano con "m" o con "g"	paperino.doc paperino.txt paperone.doc pippo.txt

Espansione delle espressioni regolari

Le espressioni regolari vengono espanse dalla shell prima di essere passate al comando:

linea di comando→espansione espressioni regolari →esecuzione comando

Ad esempio il comando "ls -l *txt" viene espanso dalla shell come "ls -l pippo.txt pluto.txt paperino.txt" e solo successivamente la linea di comando risultante viene eseguita. Il comando ls perciò verrà richiamato con quattro parametri, esattamente come se si fosse scritto il risultato dell'espansione direttamente mediante la tastiera. Il comando non dovrà mai lavorare direttamente con una espressione regolare ma solo col risultato dell'espansione.

```
ls -l *txt→ls -l pippo.txt pluto.txt
paperino.txt →esecuzione
```

Altri sistemi operativi lasciano invece al programmatore che scrive ogni singolo comando il compito di risolvere le espressioni regolari, cercandosi nella directory corrente tutti i file che soddisfano l'espressione. Ciò costringe ovviamente ad uno sforzo inutile di programmazione ed a portarsi dietro del codice che potrebbe invece essere scritto una volta per tutte.

Abbiamo già detto in una puntata precedente che, non sapendo a priori quale sarà il numero di file risultanti dall'espansione di una espressione regolare, generalmente i comandi che agiscono su file accettano sulla linea di comando un numero indefinito di argomenti. Questo fatto viene indicato nei manuali che descrivono la sintassi del comando con il simbolo "...", ad esempio "cp [options] source... directory".

Le variabili d'ambiente

La shell dispone di un proprio ambiente (environment), in cui sono definiti e mantenuti alcuni valori di stato importanti al funzionamento della shell stessa e dei programmi lanciati mediante essa (directory corrente, variabili d'ambiente, ...).

Alcune variabili vengono definite direttamente dal sistema operativo che lancia la shell oppure mediante lo script di inizializzazione /etc/profile (per ulteriori esempi si faccia riferimento al manuale della shell):

PATH contiene una lista di percorsi, separati da : (es: /bin:/usr/bin:/usr/local/bin), in cui cercare un comando inserito dall'utente. Inserendo la directory . i comandi verranno cercati anche nella directory corrente. Ciò può essere utile ma espone al rischio di eseguire dei cavalli di Troia, ad esempio il comando ls contenuto nella directory di un altro utente. Se proprio si desidera aggiungere la directory corrente almeno si faccia attenzione ad aggiungerla in coda al valore della variabile PATH, in modo che un comando venga cercato nella directory corrente solamente come ultima possibilità dopo che non è stato trovato nei percorsi di sistema.

HOME contiene il percorso della home directory dell'utente. Gli script che hanno la necessità di conoscere tale valore dovrebbero fare riferimento a questa variabile, invece di presupporre che la home dell'utente si trovi sotto la directory /home (in quanto si tratta solamente di un percorso consigliato come standard e non è detto che sia lo stesso in tutti i sistemi).

PS1, PS2 contengono la stringa che deve essere utilizzata come prompt per l'utente. Di solito contengono il valore $ e > (prompt che indica continuazione di una linea precedente). Alcune sequenze di caratteri vengono interpretate in modo particolare, ad esempio PS1="[\u@\h \W]\\$ " genera un prompt del tipo: [beppe@freddy tmp]#

TERM indica ai programmi il tipo di terminale utilizzato dall'utente (es: vt100). Utilizzando questo valore i programmi adattano il loro funzionamento in base alle caratteristiche del terminale che si sta utilizzando (ad esempio mandano il cursore a capo dopo un certo numero di colonne, utilizzano gli opportuni codici di controllo per il posizionamento del cursore, la scrittura dei caratteri in grassetto, ...). Il funzionamento è basato sull'esistenza di un database di caratteristiche dei terminali (terminfo o termcap).

LOGNAME contiene il nome di login dell'utente (es: beppe).

MAIL contiene il percorso della mailbox dell'utente. Valgono considerazioni analoghe a quelle fatte per la variabile HOME.

EDITOR personalizzando il contenuto di questa variabile è possibile dire ai programmi di utilizzare un altro editor invece di quello di default (vi).

IFS indica alla shell i caratteri da utilizzare per spezzare le parole e come separatori fra i

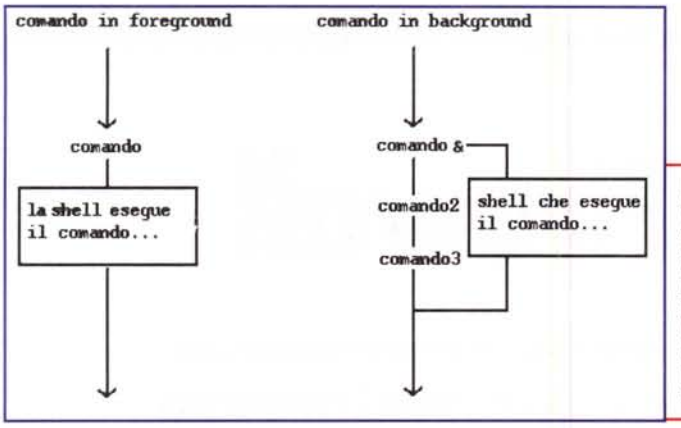

Figura 2 - Quando si inserisce una linea di comando viene lanciata una nuova shell che si occupa di gestirla. Essa eredita una fotografia delle variabili d'ambiente dalla shell padre, ma non può modificare l'ambiente di questa.

diversi argomenti. Il valore standard preassegnato è lo spazio.

Il contenuto di una variabile può essere modificato dall'utente, assegnando da linea di comando un nuovo valore mediante la scrittura:

variabile=valore

Si noti che nella linea di comando appena scritta non sono presenti spazi e che sarebbe un errore inserirne. Nel caso il valore da assegnare alla variabile contenga spazi (che verrebbero interpretati dalla shell come un separatore fra i diversi argomenti) o altri caratteri che hanno un significato particolare per la shell, è necessario quotare la stringa, ad esempio usando le virgolette:

```
variabile="il mio valore"
```

Il nuovo valore appena assegnato sarà visibile solamente alla shell in cui si è eseguita l'assegnazione. Per fare in modo che esso sia disponibile anche ai programmi da essa lanciati, è necessario "esportare" la variabile, mediante il comando:

```
variabile=valore
export variabile
```

E` possibile eseguire contemporaneamente assegnazione ed esportazione della variabile usando:

```
export variabile=valore
```

Modificando ed esportando il valore della variabile EDITOR sarà ad esempio possibile fare in modo che i programmi lanciati da questa shell utilizzino un determinato editor al posto di quello di default (generalmente /bin/vi):

```
export EDITOR=/usr/bin/joe
```

Per definire delle variabili in modo permanente è possibile inserire i comandi in /etc/profile o nel .profile personale di ogni utente. Per vedere la lista di tutte le variabili definite correntemente nell'ambiente della propria shell si utilizza il comando set (oppure printenv).

```
$ printenv
BASH=/bin/bash
BASH_VERSION=1.14.0(1)
COLUMNS=80
DISPLAY=:0.0
EUID=0
...
UID=0
WINDOWID=58720269
_=more
ignoreeof=10
```

Per accedere ad una variabile si scrive il nome facendolo precedere dal simbolo $:

```
echo $nome
andrea verdi
```

Un errore frequente è quello di dimenticarsi che il $ non deve essere indicato quando si assegna il valore alla variabile. La seguente scrittura:

```
$nome="mario rossi"
```

è errata, in quanto la shell prima di eseguire il comando tenta di sostituire la scrittura $nome con il valore della variabile nome. La linea di comando inserita viene perciò interpretata come:

```
andrea verdi=mario rossi
```

Se sono definite delle variabili che sono una il prefisso dell'altra, ad esempio NOME e NOMEUTENTE, è possibile evitare ambiguità racchiudendone il nome fra parentesi graffe:

```
echo ${NOME} ${NOMEUTENTE}
```

Analogamente a quanto abbiamo visto nel caso delle espressioni regolari, anche le variabili vengono sostituite dalla shell prima di eseguire la linea di comando:

linea di comando →sostituzione variabili con valore →esecuzione

Ad esempio:

echo Ciao $(LOGNAME →`echo Ciao beppe`→ `Ciao beppe`.

Ambito di validità dell'ambiente della shell

Il valore assegnato ad una variabile viene perduto quando la shell in cui è stato definito termina.

Infatti, quando al prompt si inserisce una linea di comando, la shell esegue (fork) un nuovo processo che si occupa di gestirlo. Questo processo è a sua volta una shell, per l'esattezza una copia esatta della shell padre, che duplica oltre al codice del programma anche la sua area dati (si veda la **Figura 2**). In questo modo la shell che gestisce i comandi inseriti eredita solamente una "fotografia" dei valori delle variabili d'ambiente della shell padre ed eventuali modifiche rimangono ristrette ad essa ed agli eventuali suoi processi figli.

Eventuali modifiche alle variabili d'ambiente da parte della nuova shell non vengono passate indietro alla shell padre (ricordate il discorso che ogni processo funziona in un ambiente separato dagli altri?). Una volta avviata la shell che sovrintende all'esecuzione di una data linea di comando, una eventuale comunicazione con la shell padre è possibile solamente mediante uno dei metodi che abbiamo analizzato la volta scorsa, oppure utilizzando l'output del comando.

L'effetto positivo di tale comportamento è che non vi è il rischio che un uso improprio delle variabili abbia ripercussioni sull'ambiente della shell padre. L'effetto negativo è invece quello che non si può, o non è così semplice come potrebbe sembrare, scrivere uno shell script che compia le stesse operazioni del comando "cd". Analogamente non è possibile inserire delle definizioni di variabili in uno script e aspettarsi che eseguendolo esse vengano rese visibili alla shell padre, in quanto tali definizioni risultano ristrette alla shell figlia che gestisce lo script.

Conclusioni

Nella prossima lezione del nostro corso introduttivo a Linux vedremo altre funzioni interessanti della bash e impareremo a realizzare semplici script mediante il linguaggio di programmazione di shell. MC