Lezione 3: Programmazione della Shell di Unix

Laboratorio di Elementi di Architettura e Sistemi Operativi

21 Marzo 2012

Parte 1: Soluzione degli esercizi

Statistiche

- Ho ricevuto 21 soluzioni
- Tutte le soluzioni hanno ricevuto voto positivo
- Le soluzioni complete e l'elenco dei voti sono disponibili sulla pagina del corso

Esercizio 7

1. *fornire il numero di file (e directory) contenuti ricorsivamente nella propria home (si può utilizzare `ls -R`? ed il comando `find`?)*

L'idea di base per risolvere l'esercizio è quella di produrre una lista dei file e directory contenuti ricorsivamente nella propria home, contando le linee in output (supponendo che ogni linea corrisponda ad un file/directory).

```
$ find .  2>/dev/null | wc -l
```

Utilizzando `ls -R` non si ottiene il conteggio esatto!

2. *elencare i file (e directory) contenuti ricorsivamente nella propria home il cui percorso relativo **non** contiene la lettera "a" o "A"*

Anche il questo caso per risolvere l'esercizio si deve prima produrre la lista dei file e directory contenuti nella propria home con il comando `find`. Si noti che il comando restituisce il percorso relativo di ogni file.

Quindi si può utilizzare `grep` per filtrare l'elenco eliminando dalla lista le righe dove appare una "a" (maiuscola o minuscola):

```
$ find .  2>/dev/null | grep -iv a
```

Parte 2: Storia dei comandi e completamento automatico

La storia dei comandi

La *storia dei comandi* è uno strumento fornito dalla shell che consente di evitare all'utente di digitare più volte gli stessi comandi:

- la storia memorizza gli ultimi 500 comandi inseriti dall'utente;
- la storia viene salvata nel file `.bash_history` al momento del logout (e riletta al momento del login);
- il comando `history` consente di visualizzare la lista dei comandi:

```
$ history | tail -4
512 ls -al
513 cd /etc
514 more passwd
515 history | tail -4
```

- ogni riga prodotta dal comando `history` è detta evento ed è preceduta dal *numero dell'evento*.
- Conoscendo il numero dell'evento che vogliamo ripetere, possiamo eseguirlo usando il metacarattere `!`:

```
$ !515
history | tail -4
513 cd /etc
514 more passwd
515 history | tail -4
516 history | tail -4
```

- Se l'evento è l'ultimo della lista è sufficiente usare `!!`:

```
$ !!
history | tail -4
514 more passwd
515 history | tail -4
516 history | tail -4
517 history | tail -4
```

- È anche possibile eseguire delle *ricerche testuali* per individuare l'evento a cui siamo interessati:

```
$ !ls
ls -al
total 491
drwxr-xr-x  16 root root      0 Oct 15 21:35 .
drwxr-xr-x  16 root root      0 Oct 15 21:35 ..
-rw-r--r--   1 root  root 87515 Jul 10 04:28 Muttrc
drwxr-xr-x   2 root root      0 Oct 15 21:27 WindowMaker
...
```

  In questo modo la shell cerca a partire dall'ultimo evento, procedendo a ritroso, un comando che inizi con `ls`.
- Racchiudendo con `?` la stringa da ricercare, la shell controllerà che quest'ultima appaia in un punto qualsiasi del comando:

```
$ !?ls?
```

Editing dei comandi

La shell mette a disposizione dell'utente dei semplici comandi di editing per facilitare la ripetizione degli eventi:

- utilizzando i tasti cursore:
  - con la freccia verso l'alto ↑ si scorre la storia dei comandi a ritroso (un passo alla volta) facendo apparire al prompt il comando corrispondente all'evento;
  - analogamente con la freccia verso il basso ↓ si scorre la storia nella direzione degli eventi più recenti.
  - le frecce sinistra ← e destra → consentono di spostare il cursore sulla linea di comando verso il punto che si vuole editare;
- le combinazioni di tasti `Ctrl-A` e `Ctrl-E` spostano il cursore, rispettivamente all'inizio ed alla fine della linea di comando;
- il tasto `Backspace` consente di cancellare il carattere alla sinistra del cursore.

Completamento dei comandi

Una caratteristica molto utile della shell è la sua abilità di tentare di completare ciò che stiamo digitando al prompt dei comandi:

```
$ pass<Tab>
```

- La pressione del tasto `<Tab>` fa in modo che la shell cerchi un comando che inizi con `pass`.
- Siccome l'unica scelta possibile è il comando `passwd`, questo sarà riportato automaticamente nel prompt.
- Se i caratteri digitati non permettono di identificare il comando, avviene quanto segue:
  - viene prodotto un suono di avvertimento al momento della pressione del tasto `<Tab>`;
  - alla seconda pressione del tasto `<Tab>` la shell visualizza una lista delle possibili alternative;
  - digitando ulteriori caratteri, alla successiva pressione del tasto `<Tab>`, la lista diminuirà fino ad individuare un unico comando.
- Oltre a poter completare i comandi, la shell bash può anche completare i nomi dei file:

```
$ tail -2 /etc/p<Tab><Tab>
passwd printcap profile
$tail -2 /etc/pa<Tab><Invio>
bianchi:fjKppCZxEvouc:500:500::/home/bianchi:/bin/bash
rossi:Yt1a4ffkGr02:501:500::/home/rossi:/bin/bash
```

- In questo caso alla prima doppia pressione del tasto `<Tab>`, la shell presenta tre possibili alternative.
- Digitando una `a` e premendo il tasto `<Tab>`, la shell può determinare in modo univoco il completamento del nome del file.

Parte 3: Variabili e ambiente

Variabili

- La shell permette di definire *variabili* per salvare dei valori
- Assegnazione: `variabile=valore`
- Per accedere al valore di una variabile, si usa l'operatore `$`
  - Esempio: se `x` vale `123`, si può usarne il valore tramite `$x`
- Per visualizzare il valore di una variabile, si usa il comando `echo`
- Per acquisire un valore da standard input: `read variabile`
- *NOTA:* I valori delle variabili sono sempre STRINGHE
- L'ouput di un comando può essere assegnato ad una variabile con l'operatore `$( )`
- Per valutazioni aritmetiche si può usare l'operatore `$(( ))`, oppure il comando `let`

Esempi

`$ x=0`	Assegna il valore 0 alla variabile `x`
`$ echo $x+1`	Mostra la stringa `0+1`
`0+1`	
`$ echo $((x+1))`	Mostra il risultato dell'operazione `x+1`
`1`	
`$ let "x+=1"`	Incrementa di 1 il valore di `x`
`$ echo $x`	Stampa il valore di `x`
`1`	
`$ read x`	Legge il valore di `x` dallo standard input
`pippo`	
`$ echo $x`	Stampa il valore di `x` sullo standard output
`pippo`	
`$ x=$(ls)`	Assegna l'output di `ls` come valore di `x`
`$ echo $x`	Stampa la lista dei file
`[ bash cat chmod cp csh date ...`	

Le variabili d'ambiente

- Le variabili sono di norma locali alla shell
- Il comando `export` consente di passare i valori delle variabili ai processi creati dalla shell (in particolare alle sub-shell)
- L'*ambiente* della shell è una lista di coppie `nome=valore` trasmessa ad ogni processo creato
- Esempio: `export PS1='\h_mionome_\w>'`
- assegna un valore a una variabile di ambiente `export variabile[=valore]`
- stampa il valore di una o tutte le variabili d'ambiente `printenv [variabile]`
- stampa il valore di tutte le variabili d'ambiente `env`

Variabili speciali

- Alcune variabili d'ambiente sono usate per memorizzare informazioni importanti:

`PWD`	percorso corrente
`SHELL`	nome della shell
`PATH`	percorsi dove cercare i comandi
`PS1`	prompt dei comandi
`HOME`	cartella home dell'utente
`USER`	nome dell'utente corrente
`HOSTNAME`	nome dell'host (computer a cui si è collegati)
`HOSTTYPE`	il tipo di architettura dell'host

Parte 4: Script e programmazione della shell

I file di comandi (script)

- È possibile memorizzare in un file una serie di comandi, eseguibili richiamando il file stesso
- Esecuzione:
  - tramite il comando `source script.sh argomenti`
    - i comandi vengono eseguiti all'interno della shell corrente
  - tramite il comando `bash script.sh argomenti`
  - eseguendo direttamente lo script: `./script.sh`
    - è necessario che il file abbia il *permesso di esecuzione*
  - Negli ultimi due casi viene lanciato un nuovo processo per il programma che deve interpretare lo script
- Per convenzione, la prima riga del file inizia con `#!`, seguita dal nome dall'interprete entro cui eseguire i comandi (`#!/bin/bash`)

Esempio

```
#!/bin/bash
date   # restituisce la data
who    # restituisce chi è connesso
```

Il carattere `#` indica che tutto quello che segue è un commento, e viene ignorato

Variabili speciali

- La bash memorizza gli argomenti della linea di comando dentro una serie di variabili speciali:

  `$1 $2 ... $9`

- Altre variabili speciali:

  `$$` PID del processo shell

  `$0` Il nome dello script/processo corrente

  `$#` il numero di argomenti

  `$* $@` tutti gli argomenti

  `$?` l'exit status dell'ultimo comando (0 se il comando è stato eseguito correttamente, un valore diverso da 0 altrimenti)

- Il comando `exit n` termina l'esecuzione dello script e assegna il valore `n` all'exit status

Esempio
numargomenti.sh

```
#!/bin/bash
echo Questo script si chiama $0
echo "E' stato eseguito con" $# argomenti
echo Il primo argomento e\' #1
echo "Il quarto argomento e'" #4
exit $#
```

```
$ ./numargomenti.sh pippo pluto paperino
Questo script si chiama ./numargomenti.sh
E' stato eseguito con 3 argomenti
Il primo argomento e' pippo
Il quarto argomento e'
$ echo $?
3
```