



DICo

Sistemi
Operativi

Bruschi
Monga

Shell

Shell
programming
Esercizi

I/O
Esercizi

Tabella
riassuntiva

Sistemi Operativi¹

Mattia Monga

Dip. di Informatica e Comunicazione
Università degli Studi di Milano, Italia

mattia.monga@unimi.it

a.a. 2011/12



DICo

Sistemi
Operativi

Bruschi
Monga

Shell

Shell
programming

Esercizi

I/O

Esercizi

Tabella
riassuntiva

Lezione IX: Shell 3

Un vero linguaggio di programmazione



DICo

Sistemi Operativi

Bruschi Monga

Shell

Shell programming

Esercizi

I/O

Esercizi

Tabella riassuntiva

La shell è un vero (Turing-completo) linguaggio di programmazione (interpretato)

- Variabili (create al primo assegnamento, uso con \$, export in un'altra shell).
 - `x="ciao"; y=2 ; /bin/echo "$x $y $x"`
- Istruzioni condizionali (valore di ritorno 0 \rightsquigarrow true)
 - `if /bin/ls piripacchio; then /bin/echo ciao; else /bin/echo buonasera; fi`
- Iterazioni su insiemi
 - `for i in a b c d e; do /bin/echo $i; done`
- Cicli
 - `/usr/bin/touch piripacchio`
 - 2 `while /bin/ls piripacchio; do`
 - 3 `/usr/bin/sleep 2`
 - 4 `/bin/echo ciao`
 - 5 `done & (/usr/bin/sleep 10 ; /bin/rm piripacchio)`



- 1 Per ciascuno dei file `dog`, `cat`, `fish` controllare se esistono nella directory `bin` (hint: usare `/bin/ls` e nel caso scrivere ‘‘Trovato’’)
- 2 Consultare il manuale (programma `/usr/bin/man`) del programma `/bin/test` (per il manuale `man expr`)
- 3 Riscrivere il primo esercizio facendo uso di `test`

Input e Output



DICo

Sistemi Operativi

Bruschi
Monga

In generale il paradigma UNIX permette alle applicazioni di fare I/O tramite:

Input

- Parametri al momento del lancio
- Variabili *d'ambiente*
- File (tutto ciò che può essere gestito con le syscall `open`, `read`, `write`, `close`)
 - Terminale (interfaccia testuale)
 - Device (per es. il mouse potrebbe essere `/dev/mouse`)
 - Rete (socket)

Output

- Valore di ritorno
- Variabili *d'ambiente*
- File (tutto ciò che può essere gestito con le syscall `open`, `read`, `write`, `close`)
 - Terminale (interfaccia testuale)
 - Device (per es. lo schermo in modalità grafica potrebbe essere `/dev/fb`)
 - Rete (socket)

Shell

Shell programming
Esercizi

I/O

Esercizi
Tabella riassuntiva

Ad ogni processo sono sempre associati tre file (già aperti)

- Standard input (Terminale, tastiera)
- Standard output (Terminale, video)
- Standard error (Terminale, video, usato per le segnalazione d'errore)

Possono essere *rediretti*

- `/usr/bin/sort < lista` Lo stdin è il file `lista`
- `/bin/ls > lista` Lo stdout è il file `lista`
- `/bin/ls piripacchio 2> lista` Lo stderr è il file `lista`
- `(echo ciao & date ; ls piripacchio) 2> errori 1>output`



Per fare esperimenti con le redirezioni può essere utile una funzione come la seguente

```
1 #include <stdio.h>
2 #include <sys/stat.h>
3 #define _POSIX_SOURCE
4 #include <limits.h>
5
6 void lsofd(void){
7     int i;
8     for (i=0; i<_POSIX_OPEN_MAX; i++){
9         struct stat buf;
10        if (fstat(i, &buf) == 0){
11            printf("fd:%d i-node: %d\n", i, buf.st_ino);
12        }
13    }
14 }
```

Shell

Shell
programming
Esercizi

I/O

Esercizi
Tabella
riassuntiva



La **pipe** è un canale, analogo ad un file, bufferizzato in cui un processo scrive e un altro legge. Con la shell è possibile collegare due processi tramite una pipe anonima.

Lo stdout del primo diventa lo stdin del secondo

```
/bin/lis | sort
```

```
ls -lR / | sort | more
```

funzionalmente equivalente a

```
ls -lR >tmp1; sort <tmp1 >tmp2; more<tmp2; rm tmp*
```

Molti programmi copiano lo stdin su stdout dopo averlo elaborato: sono detti **filtri**.



Con una pipe è possibile “collegare” lo stdout di un programma con lo stdin di un altro.

Per usare l’output di un programma sulla riga di comando di un altro programma, occorre usare la **command substitution**

```
/bin/ls -l $(/usr/bin/which sort)
```

Shell

Shell
programming
Esercizi

I/O
Esercizi

Tabella
riassuntiva



- 1 Verificare qual è il valore di ritorno di una *pipeline*, anche in caso che qualcuno dei “filtri” fallisca.
- 2 Scrivere una *pipeline* di comandi che identifichi il le informazioni sul processo floppy (`ps`, `grep`)
- 3 Scrivere una *pipeline* di comandi che identifichi il solo processo con il PPID piú alto (`ps`, `sort`, `tail`)
- 4 Ottenere il numero totale dei file contenuti nelle directory `/usr/bin` e `/var` (`ls`, `wc`, `expr`)
- 5 Si immagini di avere un file contenente il sorgente di un programma scritto in un linguaggio di programmazione in cui i commenti occupino intere righe che iniziano con il carattere `#`. Scrivere una serie di comandi per ottenere il programma senza commenti. (`grep`)
- 6 Ottenere la somma delle occupazioni dei file delle directory `/usr/bin` e `/var`



Prog. (sez. man)	Descrizione
ls (1)	list directory contents
echo (1)	display a line of text
touch (1)	change file timestamps
sleep (1)	delay for a specified amount of time
rm (1)	remove files or directories
cat (1)	concatenate files and print on the standard output
man (1)	an interface to the on-line reference manuals
test (1)	check file types and compare values
sort (1)	sort lines of text files
date (1)	print or set the system date and time
more (1)	file perusal filter for crt viewing
which (1)	locate a command
ps (1)	report a snapshot of the current processes.
tail (1)	output the last part of files
wc (1)	print the number of newlines, words, and bytes in files
bc (1)	An arbitrary precision calculator language
grep (1)	print lines matching a pattern
cut (1)	remove sections from each line of files
du (1)	print disk usage



- “A Brief Introduction to Unix (With Emphasis on the Unix Philosophy)”, Corey Satten <http://staff.washington.edu/corey/unix-intro.pdf>
- http://en.wikipedia.org/wiki/Unix_philosophy
- “The UNIX Time-Sharing System”, Ritchie; Thompson <http://www.cs.berkeley.edu/~brewer/cs262/unix.pdf>