Come funziona Linux:
funzioni avanzate della shell e comandi utili per la programmazione di script

Dopo avere introdotto i concetti base della shell Bash, in questa lezione del nostro corso su Linux vedremo ulteriori utilizzi della stessa ed inizieremo ad imparare come usarla per realizzare programmi, anche relativamente complessi.

Quinta parte

di Giuseppe Zanetti

Riepilogo dei comandi di ridirezione dell'I/O

Abbiamo già incontrato nelle puntate precedenti gli operatori di input/output. Prima di procedere è bene fare un breve riepilogo e completare alcuni punti importanti. Una delle cose che rendono così potente l'utilizzo della linea di comando in un sistema UNIX è la possibilità di scrivere programmi indipendenti dai file o dai dispositivi in cui essi devono scrivere e leggere i dati.

Ad un programma sono infatti associati tre file: standard input, standard output e standard error, da cui vengono prelevati i dati inseriti dall'utente e mandati i risultati delle elaborazioni e gli eventuali errori. Se non altrimenti specificato, quando si esegue un programma dalla linea di comando, a tutti questi file viene associato il terminale su cui l'utente sta lavorando (a cui corrisponde un file speciale, ad esempio /dev/tty1 per la prima console virtuale oppure /dev/ttyS1 per un terminale seriale o /dev/ttyp1 per una sessione di lavoro mediante telnet).

Ciò permette di interagire col programma, inserendo i dati necessari da tastiera e leggendo output ed errori direttamente sullo schermo. Non in tutti i casi i programmi richiedono l'interazione con l'utente ed esistono delle situazioni in cui è desiderabile utilizzare dei normali file per ricevere l'input e per salvare l'output e gli eventuali errori generati. La bash permette di ridirigere i file associati ad un processo in modo molto semplice, mediante i seguenti operatori:

Ridirezione dell'input di un programma:

```
mioprogramma <input.txt
```

Ridirezione dell'output di un programma:

```
mioprogramma >output.txt
```

Ridirezione degli errori generati dal programma:

```
mioprogramma 2>errori.txt
```

Non volendo sovrascrivere ogni volta che si riesegue il programma il file di output, si può utilizzare l'operatore >>, il quale invece di cancellare il file accoda l'output senza perdere il contenuto precedente:

```
mioprogramma >>output.txt
```

Ovviamente è possibile ridirezionare contemporaneamente anche più file per volta, come in questo caso:

```
mioprogramma <input.txt >output.txt 2>errori.txt
```

Se il programma che si sta eseguendo necessita di un certo tempo per l'elaborazione, può essere utile lasciarlo lavorare tranquillamente in background:

```
mioprogramma <input.txt >output.txt 2>errori.txt &
```

Volendo mandare in un unico file sia output che errori, è possibile ridirezionare il file degli errori su quello di output (o viceversa) e ridirezionare in modo opportuno il risultato:

```
mioprogramma <input.txt 2>&1 >output_e_errori.txt
```

Tentiamo di capire più a fondo il meccanismo: ai file di input, output ed errori associati al processo corrispondono in linguaggio C i "descrittori" di file numerati come 0, 1 e 2. Da shell è possibile utilizzare questi indici per ridirezionare i file: la scrittura "n>" indica la ridirezione in scrittura del file con indice *n*, mentre "&n" indica la ridirezione in lettura dello

stesso file. Nel caso delle scritture "2>" e "&1" l'indice utilizzato viene espresso in modo esplicito, mentre scrivendo "<" e ">", gli indici 0 e 1 sono sottointesi (le stesse operazioni potrebbero essere riscritte anche come "0<" e "1>").

Non necessariamente i file verso e da cui si ridirezionano input e output devono essere file normali, ma è anche possibile utilizzare la ridirezione da e verso file speciali che rappresentano periferiche hardware (vi ricordate che in UNIX ogni periferica ha associato uno o più file in /dev?). In questo modo è possibile ad esempio mandare l'uscita di un programma direttamente sulla stampante:

```
mioprogramma >/dev/lp0
```

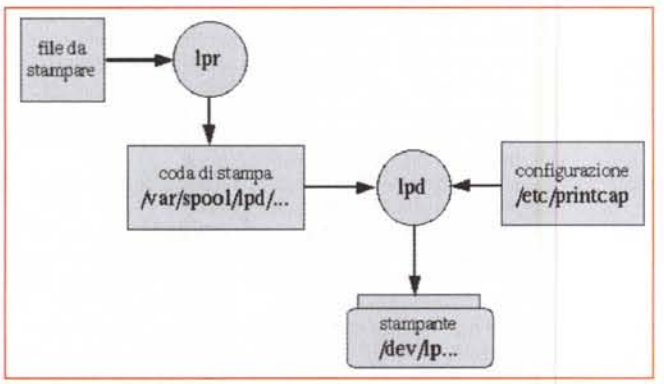


Figura 1 - Funzionamento dello "spooler" di stampa lpd (figura tratta dal libro AppuntiLinux di Daniele Giacomini, disponibile su http://www.pluto.linux.it/ildp).

oppure utilizzare una periferica hardware o un file speciale come input:

```
cat /dev/audio >nota.au
cat nota.au >/dev/audio
```

L'esempio precedente mostra come registrare "al volo" e come riascoltare un messaggio vocale senza utilizzare un apposito programma. Il formato generato e accettato del file speciale /dev/audio è il μ-law a 8 kHz. Volendo utilizzare un formato compatibile con .wav si deve invece utilizzare il file speciale /dev/dsp:

```
cat /dev/dsp >nota.wav
cat nota.wav >/dev/dsp
```

Apriamo una parentesi: volendo generare o registrare un effetto sonoro dall'interno di un proprio programma, ad esempio scritto in linguaggio C, non è necessario utilizzare librerie particolari, ma è sufficiente leggere o scrivere i campioni che compongono il suono su /dev/audio:

```
int audio;
...
audio=open("/dev/audio",O_WRONLY);
...
r=write(audio,s,1);
```

Ad esempio per "fare suonare" un file .au preregistrato, sarà sufficiente scrivere un ciclo che prelevi i dati da questo file e li scriva con write sul file "audio". Non è necessario preoccuparsi delle temporizzazioni, in quanto della loro gestione si occupa il device driver che sovrintende a /dev/audio. Il nostro programma dovrà fare solamente attenzione a fornire i dati con un ritmo di almeno 8000 campioni al secondo, pena il sentire delle pause o dei fastidiosi "click" nel suono in uscita. Sapendo questo diventa semplicissimo scrivere un semplice generatore di onde. Ad esempio per ottenere un'onda quadra a 4000 Hz sarà sufficiente alternare su /dev/audio la scrittura dei valori 0 e 255. Volendo ottenere una sinusoide occorrerà calcolare i valori di un certo numero di campioni utilizzando la funzione matematica sin(). Per risparmiare potenza di calcolo, i conti possono essere eseguiti una sola volta ed inseriti in un vettore da cui possano in seguito essere riletti molto velocemente.

Utilizzo dell'operatore pipe

Il metodo di stampa visto in precedenza è abbastanza spartano, in quanto i dati vengono mandati direttamente all'hardware senza nessun controllo sul fatto che un altro utente o programma stia già stampando. Un metodo più pulito è quello di utilizzare l'operatore di pipe per stampare utilizzando l'apposito comando lpr, il quale invia la stampa in una apposita coda che viene gestita dallo "spooler" lpd (**figura 1**):

```
mioprogramma | lpr
```

L'operatore "pipe", letteralmente "tubo", permette di utilizzare l'output di un programma direttamente come input per un altro programma. Un metodo alternativo per ottenere il medesimo risultato, inutile ai fini pratici ma che in qualche modo rende l'idea di come potrebbero funzionare le cose, è quello di utilizzare la seguente sequenza di comandi:

```
mioprogramma >/tmp/pippo
lpr </tmp/pippo
rm /tmp/pippo
```

In realtà, al contrario di MS-DOS, il quale, essendo un sistema monotasking, usa il metodo del file temporaneo per implementare l'operazione di pipe, nei sistemi derivati da UNIX viene creata un'apposita coda per lo scambio dei dati

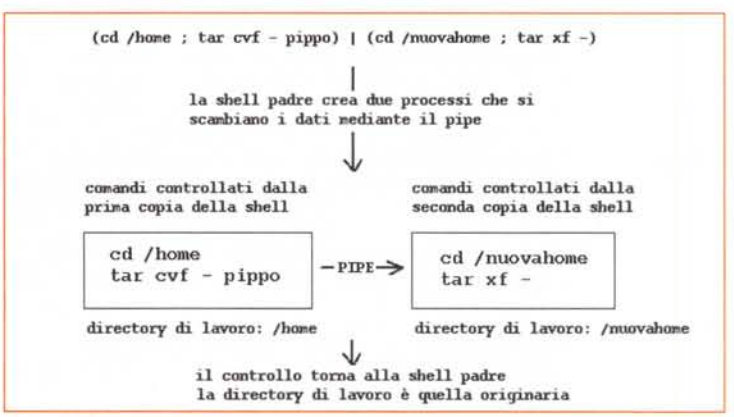


Figura 2 - Flusso delle operazioni nel caso vengano utilizzate le parentesi da shell.

fra due processi in esecuzione direttamente nella memoria della macchina.

Concatenando fra loro più operazioni di pipe è possibile realizzare una "pipeline" ("linea di tubi"), comprendente anche più di due comandi da eseguire in cascata. Ogni comando della pipeline viene eseguito dalla shell come un processo separato e l'operatore di pipe implementa una forma di comunicazione fra processi (pur con la limitazione di essere unidirezionale).

I filtri

Grazie al meccanismo appena descritto si possono compiere operazioni anche complesse utilizzando opportunamente solo un ristretto numero di programmi e comandi che compiono operazioni molto semplici. In particolare esiste tutta una serie di programmi standard che compiono operazioni anche molto semplici sui file di testo e che possono essere utilizzati in cascata (per capire gli esempi che seguono si faccia riferimento alla descrizione dei comandi presenti nel pac-

Comandi utili

GNU textutils

Si tratta di una raccolta di programmi, previsti dallo standard POSIX e presenti in tutti i sistemi UNIX, i quali possono essere utilizzati (direttamente su uno o più file oppure come filtri) per elaborare il contenuto di file di testo. Generalmente si tratta di comandi che compiono operazioni abbastanza semplici ma che hanno il pregio di essere utilizzabili in cascata per ottenere risultati complessi. Rispetto ai comandi originali di UNIX, la versione scritta nell'ambito del progetto GNU - disponibile "di serie" su tutti i sistemi Linux - offre delle caratteristiche e funzioni aggiuntive interessanti. I dettagli d'uso dei diversi comandi presenti nel pacchetto possono essere studiati sui manuali in linea (es.: "man grep").

cat	concatena più file in un unico output. Utilizzato con un solo parametro manda in output il contenuto di un file
cksum	analogo a sum
comm	confronta due file linea per linea e propone in output le linee presenti in un file ed assenti nell'altro oppure quelle presenti in entrambi i file
csplit	suddivide un file o l'input in più file, spezzandolo in corrispondenza di regole definite dall'utente
cut	permette di tagliare pezzi delle righe in input. La selezione può essere fatta specificando intervalli di colonne (es.: cut -c8-20 file.txt) oppure "campi" separati da un carattere (es.: cut -f5 -d: /etc/passwd)
expand	converte il carattere di tabulazione TAB in una sequenza di spazi
fmt	formatta un file o lo standard input in paragrafi, spezzando le linee in modo che non eccedano una data lunghezza
fold	spezza le linee in un file o nello standard input e le "giustifica" ad una data lunghezza
head	restituisce le prime n righe di un file
join	a partire da due file, accoppia le linee in essi contenute in base ad una chiave comune ed unisce il risultato
md5sum	analogo a sum: verifica il checksum di un file utilizzando l'algoritmo crittografico MD5
nl	numera le linee in un file, eventualmente separando e tenendo conto in modo opportuno di pagine e paragrafi
od	visualizza il contenuto di un file o lo standard input in ottale, esadecimale o in altri formati non-ASCII
paste	unisce due file. Può essere utilizzato per ottenere un documento a due colonne a partire dai file contenenti separatamente le due colonne: per prima cosa è necessario giustificare le due colonne utilizzando il comando "fold -35 colonna1.txt >colonna1ok.txt". In seguito i file risultanti possono essere uniti mediante: "paste colonna1ok.txt colonna2ok.txt"
pr	prepara un file o lo standard input per la stampa, formattando opportunamente i paragrafi e aggiungendo in testa ad ogni pagina una riga col nome del file e la numerazione progressiva delle pagine
sort	ordina alfabeticamente le righe in input. E' possibile scegliere di utilizzare l'ordine inverso (-r) oppure di considerare eventuali cifre ad inizio riga col loro valore numerico (-n) invece che con quello letterale
split	spezza un file o lo standard input in più file di una data dimensione o con un dato numero di linee
sum	ottiene un checksum del file, utile per verificare che non sia stato modificato (da estranei oppure a seguito della trasmissione dello stesso)
tac	concatena più file e ne stampa il contenuto in ordine inverso (dall'ultima riga alla prima)
tail	restituisce le ultime n righe di un file. Può essere utilizzato, mediante l'opzione -f, per tenere sotto controllo in tempo reale eventuali cambiamenti in un file (es.: tail -f /var/log/messages)
tr	dati due insiemi di caratteri, sostituisce tutte le occorrenze di un carattere contenuto nel primo insieme nel corrispettivo nella stessa posizione nel secondo insieme. E' possibile specificare intervalli di caratteri utilizzando la scrittura [x-y] già vista per le espressioni regolari (es.: [A-Z] per selezionare tutte le lettere maiuscole oppure "[a-z]" per le minuscole)
tsort	esegue un ordinamento "topologico" su un file
unexpand	Esegue l'operazione inversa di expand, ovvero converte sequenze di spazi in caratteri di tabulazione
uniq	elimina eventuali righe ripetute nel file di input, che deve essere ordinato alfabeticamente (ad esempio mediante sort: sort file | uniq)
wc	conta il numero di parole (opzione -w), linee (-l) o caratteri (-c) contenuti in un file

Altri comandi

Molto utili per trattare file di testo o per essere utilizzati negli script sono anche i seguenti comandi:

grep	permette di ricercare espressioni regolari all'interno di un file, scrivendo in output tutte le linee in cui compare la stringa specificata. Mediante opportuni parametri è possibile eseguire ricerche insensibili alla differenza fra maiuscole e minuscole (opzione -i), visualizzare tutte le righe che non contengono la stringa (-v), oppure scrivere solamente i nomi dei file in cui la stringa viene trovata
find	trova, all'interno del filesystem, tutti i file che soddisfano determinate regole (data di creazione, dimensione, permessi, nomi, ...) e permette di eseguire un comando per ogni file trovato.
sed	permette di eseguire operazioni complesse sull'input, ad esempio sostituzioni di stringhe di caratteri che soddisfano una data espressione regolare, cancellazione di righe o parole, ...
awk	si tratta di un vero e proprio linguaggio di programmazione che permette di eseguire operazioni molto complesse su un file in input

chetto GNU textutils, che si può trovare nel **riquadro 1**).

Spesso tali programmi vengono utilizzati come "filtri". Essi filtrano lo standard input che viene loro passato, restituendo in uscita un risultato che può eventualmente essere inviato ad un altro filtro per un'ulteriore elaborazione.

Molti comandi possono essere utilizzati come filtri semplicemente omettendo nella linea di comando i parametri che specificano i file su cui essi devono agire. Ad esempio il comando "more nomefile ..." visualizza uno o più file sullo schermo, suddividendo l'output in pagine e attendendo la pressione di un tasto per continuare la visualizzazione. Utilizzato senza parametri esso visualizza ciò che viene passato nello standard input, ovvero i risultati di elaborazioni precedenti nel caso si utilizzi un pipe:

```
ls -la | more
```

Alcuni comandi invece per funzionare come filtri prevedono che venga passato loro come parametro il valore "-", che in questo modo identifica lo standard input:

```
cat file1.txt - file2.txt
```

Il seguente esempio ricava dalla lista degli utenti del sistema, /etc/passwd, il quinto campo (gecos), elimina eventuali righe vuote (il simbolo ^ in una espressione regolare identifica l'inizio riga, mentre $ indica il fine riga), trasforma le minuscole in maiuscole, ordina alfabeticamente le righe così ottenute, eliminando quelle ripetute e presentando il risultato finale a pagine di 20 caratteri alla volta:

```
cut -f5 -d: /etc/passwd | grep -v ^$ | tr
"[a-z]" "[A-Z]" | sort | uniq | more -20
```

La filosofia che sta alla base dell'utilizzo delle pipeline è quella di scomporre il problema generale in tanti piccoli pezzettini. Una volta fatta un po' di esperienza sui comandi fondamentali ci si accorgerà che disponendo in modo diverso pochissimi mattoni è possibile risolvere molto semplicemente anche problemi a prima vista molto complessi.

Utilizzo delle parentesi

I comandi inseriti in una linea di comando vengono eseguiti in modo sequenziale. Ad esempio nella seguente linea di comando:

```
echo ciao ; echo mondo | tr [a-z] [A-Z]
```

viene eseguito dalla stessa shell per prima il comando "echo ciao" e solo successivamente la pipeline "echo mondo l tr [a-z] [A-Z]" (che ha l'effetto di convertire tutte le lettere minuscole in maiuscole). Il risultato è perciò il seguente:

```
ciao
MONDO
```

Utilizzando le parentesi, è possibile raggruppare più comandi in modo che vengano eseguiti assieme. Se proviamo a raggruppare i due comandi echo mediante le parentesi:

```
(echo ciao ; echo mondo ) | tr "[a-z]"
"[A-Z]"
```

otterremo il seguente risultato:

```
CIAO
MONDO
```

In questo caso infatti viene passato a tr tutto l'output generato dai comandi contenuti all'interno delle parentesi.

E' bene ricordare che ogni shell è una scatola a sé e che le variazioni introdotte non vengono passate indietro alla shell padre. Nel seguente esempio l'effetto di cambio della directory di lavoro dovuto al comando "cd", funziona solamente all'interno delle relative parentesi:

```
(cd /home ; tar cf - pippo ) | (cd /nuo-
vahome ; tar xf -)
```

il risultato è quello di lanciare due processi, ognuno gestito da una diversa copia della shell, che lavorano su directory diverse e che si parlano fra loro mediante il pipe (il flusso delle operazioni è rappresentato in **figura 2**). Ogni shell esegue in modo sequenziale il relativo "cd" e una copia del programma archiviatore "tar".

Il primo tar (che lavora nella directory /home) crea un archivio contenente tutti i file nella directory pippo e nelle sue sottodirectory e lo manda sullo standard output (identificato dal simbolo "-"). Il secondo tar (che lavora nella directory /nuovahome) riceve dallo standard input (simbolo "-") l'archivio e lo scompatta. Il risultato finale è quello di spostare tutti i file contenuti nell'home directory dell'utente pippo da /home/pippo a /nuovahome/pippo.

La sostituzione di un comando

Racchiudendo una linea di comando fra una coppia di apici inversi (carattere `, da non confondere col simbolo ' di apostrofo o singolo apice) è possibile fare in modo che i comandi in essa contenuti vengano eseguiti (da una copia della shell appositamente lanciata) ed il risultato inserito nel punto in cui si trovava la parte di linea di comando racchiusa fra apici:

```
comando1 `comando2` Æ comando output di co-
mando2 Æ esecuzione
```

Un esempio aiuterà a capire meglio di ogni spiegazione:

```
$ echo Sono le ore `date +%H` e `date +%M`
minuti
```

I due comandi racchiusi fra apici inversi vengono sostituiti col risultato degli stessi, rispettivamente le due cifre che identificano l'ora ed il minuto correnti:

```
echo Sono le ore `date +%H` Æ echo Sono le
ore 20
```

Il risultato finale è perciò il seguente:

```
Sono le ore 20 e 42 minuti
```

Un altro utilizzo interessante è quello di inserire in un documento di testo una lista di file e di eseguire un comando simile a:

Il comando find

Uno dei comandi più interessanti per l'amministrazione di un sistema Linux è senz'altro find: esso permette di ricercare ricorsivamente in una directory i file che verificano determinate condizioni e di eseguire su di essi delle operazioni. La sintassi da utilizzare è la seguente, in cui si specificano la directory da cui iniziare la ricerca, le condizioni da verificare e l'azione da intraprendere per ognuno dei file trovati:

```
find directory -condizioni -azione
```

Le condizioni di ricerca più utilizzate sono le seguenti:

-mmin n	vengono cercati tutti i file modificati da esatta mente n minuti
-mmin +n	vengono cercati i file modificati da più di n minuti
-mmin -n	vengono cercati i file modificati da meno di n minuti
-user user	vengono cercati i file appartenenti ad un dato utente. E` possibile specificare sia lo UID che il nome di login dell'utente desiderato
-group group	vengono cercati i file appartenenti ad un dato gruppo, anche in questo caso si possono utilizza re sia il GID che il nome simbolico
-name expr	vengono cercati i file il cui nome verifica l'espres sione regolare expr. Nella valutazione dell'espres sione viene tenuto in considerazione solamente il nome del file e non l'intero path
-regex expr	verranno ricercati i file il cui path verifica l'espres sione regolare expr. Si noti la differenza col caso precedente
-perm perm	vengono cercati tutti i file in cui tutti i permessi coincidono esattamente con quelli proposti, che possono venir forniti sia in forma ottale (es.: 755) che simbolica (es.: -rwxr-xr-x)
-perm -perm	vengono ricercati i file in cui sono abilitati tutti i permessi che nella maschera perm, vista come numero binario, sono posti ad uno
-perm +perm	vengono cercati i file in cui anche solo alcuni dei permessi coincidano con quelli proposti. Essi ven gono confrontati come se fossero un numero binario.
-type tipo	vengono ricercati solamente file di un certo tipo: f file ordinari d directory l link simbolici b file speciali gestiti a blocchi c file speciali gestiti a carattere

Se non viene esplicitamente specificata una azione da intraprendere sui file trovati, verrà utilizzata -print. Le azioni maggiormente utilizzate sono le seguenti:

-print	scrive in output il path completo dei file trovati
-fprint	analoga a -print, con la differenza che il risultato vie ne scritto in un file
-exec comando \;	per ogni file trovato viene eseguito il comando specificato. Il nome del file che di volta in volta verrà utilizzato, viene indicato col simbolo {}. Il co mando deve essere terminato con il simbolo \;
-ok comando \;	analogo al precedente, con la differenza che viene richiesta conferma prima di eseguire l'operazione

Nel seguente esempio il comando find viene utilizzato per cancellare i file core.nomeprogramma presenti nel sistema. Essi vengono creati da Linux quando un programma termina in modo anomalo, ad esempio per un tentativo di scrittura fuori dall'area di memoria riservata al programma. L'operazione di cancellazione deve essere eseguita in modo interattivo e sotto il controllo dell'utente, in quanto potrebbero esistere altri file col nome che verifica la condizione ma che non sono dei dump:

```
# find / -name "core.*" -ok rm {} \;
< rm ... /usr/Icons-0.1/small/core.xpm > ? n
< rm ... /home/beppe/core.myprogram > ? y
```

Si è fatto uso del quoting per evitare che l'espressione * venga interpretata dalla shell prima che da find.

Nel prossimo esempio useremo find per cercare tutti i file che hanno il nome che termina con l'estensione .gif (sia maiuscolo che minuscolo) presenti nella directory dell'utente pippo e di ogni file trovato faremo una copia in /tmp/gif:

```
# find /home/pippo -name "*.[Gg][Ii][Ff]" -
exec cp {} /tmp/gif \;
```

I seguenti esempi mostrano invece le differenze nell'utilizzo della funzione -perm. Per trovare tutti i file con esattamente i permessi rwxrwxrwx (777) si utilizza un comando simile al seguente:

```
# find / -perm 777
```

Per ricercare invece i file che abbiano il solo permesso di scrittura accessibile a tutti gli utenti (??????w?) useremo invece:

```
# find / -perm -002
```

Infine vediamo come cercare tutti i file normali (non directory o file speciali) con estensione .txt modificati nell'ultima ora. I file trovati verranno aggiunti ad un archivio in formato tar:

```
# find / -type f -name "*.txt" -mmin -60 -
exec tar -rvf archivio.tar {} \;
```

```
more `cat lista.txt`
```

La parte `cat lista.txt` viene sostituita col contenuto di lista.txt:

```
more `lista.txt` Æ more pippo.jpg pluto.jpg
paperino.jpg
```

Tale operazione può essere utilizzata ad esempio per creare mediante tar un archivio di un insieme selezionato di file:

```
tar cvf backup.24.03.2000.tar `cat file.im-
portanti.txt`
```

Invece di una lista precompilata è possibile generare "al volo" l'elenco di file mediante il comando find (vedere **riquadro 2**). Il seguente esempio mostra come realizzare un backup incrementale dei soli file modificati dopo l'ultimo backup:

```
tar cvf backup.tar `find . -newer .last`
touch .last
```

L'opzione -newer di find fa in modo che find trovi solamen-

te i file modificati più recentemente del file .last, che viene utilizzato per tener traccia della data dell'ultimo backup. Dopo aver creato il nuovo archivio, il file viene "toccato" mediante il comando touch, in modo da aggiornarne la data di ultima modifica.

Il linguaggio della shell

La shell mette a disposizione alcune strutture utili per controllare il flusso dell'elaborazione. Esse sono del tutto simili a quelle presenti nei comuni linguaggi di programmazione (if...then...else, while...do...done, for...do..done, ...) e possono essere usate all'interno degli script oppure inserite direttamente nella linea di comando.

Il seguente esempio mostra come eseguire un file dipendentemente dal valore ritornato in uscita di un programma. Per convenzione i programmi UNIX ritornano nella variabile $? il valore 0 nel caso l'elaborazione sia terminata correttamente. Nel caso siano avvenuti degli errori di solito viene ritornato invece un valore diverso da zero.

I valori di uscita possono essere utilizzati da alcuni programmi anche per segnalare alla shell situazioni particolari, ad esempio grep ritorna 0 se trova almeno una occorrenza della espressione regolare cercata oppure 1 nel caso non ne venga trovata nessuna. Per maggiori dettagli si faccia al solito riferimento al manuale in linea del comando che interessa. Detto ciò, scriviamo, direttamente sulla linea di comando, un semplice if che controlli se un utente è presente nel sistema:

```
grep -q ^beppe$ /etc/passwd
if [ $? = 0 ]
> then
>     echo "L\'utente beppe esiste"
> else
>     echo "L\'utente beppe non esiste"
> fi
```

Il prompt > (definito nella variabile d'ambiente PS2) viene generato dalla shell e indica che essa è in attesa della "chiusura" di un comando precedente, nel nostro caso if. Nel comando if è necessario porre una certa attenzione all'utilizzo della spaziatura. In realtà il simbolo [è un alias del comando test, perciò la riga precedente può anche essere riscritta come:

```
if test $? = 0
```

Volendo testare un valore diverso da zero è possibile scrivere:

```
if [ ! $? = 0 ]
```

Volendo creare uno script, si devono inserire, mediante un editor, i comandi appena visti in un file, ad esempio testbeppe.sh (l'estensione non è necessaria), facendoli precedere, come abbiamo visto la volta scorsa, dalla linea che indica al sistema operativo che si tratta di uno shell script:

```
#!/bin/sh

grep -q ^beppe$ /etc/passwd
if [ $? = 0 ]
then
    echo "L\'utente beppe esiste"
else
    echo "L\'utente beppe non esiste"
fi
```

A questo punto, prima di poter eseguire il file, si deve assegnare opportunamente il permesso relativo all'eseguibilità da parte dell'utente proprietario:

```
$ chown u+x testbeppe.sh
$ ./testbeppe.sh
L'utente beppe esiste
```

Volendo è possibile parametrizzare lo script sostituendo in ogni posto dove compare la scritta "beppe" il simbolo ${1}, che identifica il primo parametro passato nella linea di comando. Approfittiamo della modifica anche per fare ritornare alla shell chiamante un valore che indichi se la ricerca è andata a buon fine. Per far ciò salveremo in una variabile il valore ritornato da grep e lo ritorneremo a nostra volta. Lo script perciò diventa:

```
#!/bin/sh

grep -q ^${1}$ /etc/passwd
R=$?

if [ $R = 0 ]
then
```

Il comando sed

Vediamo i due esempi più comuni di utilizzo del comando sed. Esso può essere utile per sostituire una espressione regolare con un'altra all'interno di un file:

```
sed "s/expr1/exp2/g"
```

ad esempio:

```
$ sed "s/Srl/SpA/g" <ditta.txt
```

Al posto del carattere / è possibile utilizzare come separatore un altro carattere. Ciò risulta particolarmente utile nel caso si sostituiscano nomi di percorsi per evitare di dover quotare ogni occorrenza del simbolo /. Invece di:

```
$ sed "s/\/etc/\/usr\/local\/etc/g" <file.txt
```

risulta infatti più conveniente scrivere:

```
$ sed "s,/etc,/usr/local/etc,g" <file.txt
```

Omettendo la stringa da sostituire si ottiene il risultato di cancellare tutte le occorrenze di una data espressione regolare:

```
$ sed "s/[Pp][Aa][Ss][Ww][Oo][Rr][Dd]//g"
```

lo stesso risultato lo si può ottenere utilizzando il comando:

```
sed "s/exp1/d"
```

Quelli appena visti sono solamente alcuni semplici esempi di utilizzo del comando sed. Altri possono essere reperiti nel manuale in linea.

```
      echo "L\'utente ${1} esiste"
else
      echo "L\'utente ${1} non esiste"
fi

exit $R
```

Sarebbe sbagliato lasciare dentro all'if il valore $?, in quanto il valore di ritorno controllato non deriverebbe più dal grep, bensì dal comando di assegnazione.

I metacaratteri del tipo ${n} (esempio: ${1} e ${2}) vengono espansi dalla shell con i parametri passati nella posizione corrispondente nella linea di comando. Le parentesi graffe non sarebbero di per sé obbligatorie e le scritture $1 e $2 funzionano altrettanto bene di ${1} e ${2}.

E' bene tuttavia ricordarsi comunque di mettere le parentesi graffe, per evitare cattive abitudini, che porterebbero ad errori tipo $11 (che è ambiguo, e viene riconosciuto dalla shell come $1 seguito dal carattere "1"). Molto meglio scrivere ${1}, ${2}, ..., ${10}, ${11}.

Il segreto di Linus

Transmeta annuncia un nuovo chip rivoluzionario

di Giuseppe Zanetti

Quando Linus Torvalds dalla Finlandia accettò di spostarsi a lavorare negli Stati Uniti, in molti si chiesero quale progetto si nascondesse dietro il nome di Transmeta, azienda californiana che si occupava di "Alternative VLSI engines for multimedia PCs", al cui capitale partecipava anche Paul Allen, fondatore assieme a Bill Gates di Microsoft, così come riportato sul sito http://www.paulallen.com/business/investments/. L'URL della ditta di Santa Clara per quasi quattro anni e mezzo non ha fornito ulteriori informazioni, limitandosi ad indirizzare i curiosi verso una pagina con scritto un enigmatico "sito in preparazione", tanto che più di qualcuno ad un certo punto ha temuto che veramente Linus avesse abbandonato il "suo" sistema operativo per mettersi a lavorare per il nemico. Solo recentemente sulla homepage aziendale ha iniziato a comparire un nome, Crusoe, con la promessa che i dettagli sarebbero stati resi disponibili entro poco tempo e che si sarebbe trattato di una notizia rivoluzionaria al punto tale da cambiare le carte in gioco nel mercato dei computer.

Il processore Crusoe

Crusoe (http://www.crusoe.com/) è il primo processore in cui il set di istruzioni è implementato interamente come software. Non bastasse ciò, è anche il primo processore ad offrire contemporaneamente le seguenti caratteristiche:

- consumi estremamente limitati
- performance elevate
- compatibilità con il mondo x86.

Il team di ingegneri di Transmeta durante la presentazione di Crusoe. Si noti al centro la presenza di Linus Torvalds.

Si tratta di un chip very-long-instruction-word (VLIW) a 128 bit in cui ogni istruzione della macchina non è cablata direttamente in hardware, ma viene convertita al volo da un apposito programma, detto "Code Morpher Software" (CMS).

Questo approccio permette di emulare praticamente qualunque altra piattaforma. Attualmente entrambi i chip in produzione offrono, dal punto di vista dell'utente, un set di comandi compatibile con la piattaforma x86 e l'emulazione di parte del chipset, pur essendo in realtà due CPU completamente diverse, addirittura dotate di set di comandi diversi e incompatibili fra loro.

Il primo chip introdotto da Transmeta è il TM3120, progettato per applicazioni economiche di Internet mobile, come Web pad, telefoni cellulari e palmtop computer. Esso consuma solamente 1 W a 400 MHz e dispone di 108 Kb di cache. Il sistema operativo consigliato è una versione "mobile" di Linux (ecco scoperto cosa ci stava a fare Linus in Transmeta).

La versione high-end del processore (TM5400) offre invece frequenze di lavoro di 500 o 700 MHz (sempre consumando 1 solo watt!), dispone di 400 Kb di cache ed è pensato per essere utilizzato per realizzare notebook leggeri e a basso consumo. I sistemi operativi consigliati sono in questo caso Linux, Windows 2000 e NT.

Il costo del processore TM5400 varia fra i 119$ della versione a 500 MHz e i 329$ di quella a 700 MHz. Il TM3120 risulta notevolmente più economico: soli 65$ per la versione "piccola" a 333 MHz e 89$ per i 400 MHz. I prezzi sono in linea, se non addirittura inferiori, rispetto alla concorrenza. Essendo la gran parte del lavoro svolta dal software, il design del chip ne risulta estremamente semplificato e la sua logica contiene solamente un quarto dei transistor presenti in un Pentium III. Ciò significa notevoli potenzialità di risparmio nei processi di fabbricazione del processore, oltre che un minor consumo di energia e in tempi di test dell'hardware più ridotti.

La traslazione del codice x86 nel set di istruzioni reali della macchina (che è diverso per il TM3120 e per il TM5400!) avviene al momento del caricamento in memoria del codice. Vengono tradotti contemporaneamente gruppi formati da più istruzioni, in modo da ottimizzare la qualità del codice generato (ogni gruppo di quattro istruzioni x86 viene codificato in una sola, velocissima, istruzione di Crusoe). Tale approccio semplifica la CPU da tutti i meccanismi di ottimizzazione e riorganizzazione delle istruzioni che permettono l'esecuzione simultanea di più istruzioni nelle macchine x86 tradizionali senza che vi sia un degrado significativo nelle performance. Una volta che il codice è stato tradotto, vengono utilizzati dei meccanismi interni di caching allo scopo di evitare il più possibile di ripetere l'operazione per gruppi uguali di istruzioni. Ciò è particolarmente utile nel caso il programma esegua dei cicli ripetuti. L'impatto della traslazione è perciò significativo solamente al momento del primo caricamento, ma poi la macchina offre prestazioni paragonabili ad una CPU x86 (circa il 30% più lento a parità di clock).

Il risparmio di energia

Una delle caratteristiche più interessanti della nuova CPU è la presenza della tecnologia LongRun, che permette di risparmiare energia, adeguando la frequenza di funzionamento ed il voltaggio di

Per chi non ricordasse il discorso fatto la volta precedente riguardo all'espansione delle espressioni regolari da parte della shell, ricordo che la sostituzione avviene prima di passare i parametri al programma. Ad esempio:

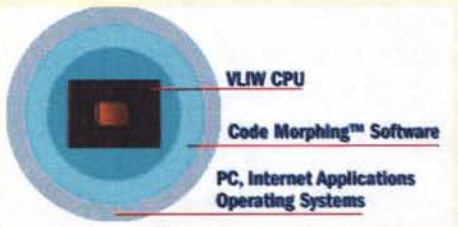
```
./mioscript.sh *jpeg Æ ./mioscript.sh pippo.jpeg pluto.jpeg minni.jpeg
```

I parametri passati saranno perciò accessibili allo script nel seguente modo:

```
${1} pippo.jpeg
${2} pluto.jpeg
${3} minni.jpg
```

Conclusioni

Nella prossima puntata continueremo a studiare le strutture e le funzioni di programmazione messe a disposizione dalla shell di Linux, mediante l'analisi di shell script. MC



Schema di funzionamento del Code Morpher.

alimentazione del processore al carico istantaneo di lavoro. Al contrario del sistema SpeedStep presente nei nuovi processori di Intel - presentato esattamente un giorno prima rispetto a LongRun - che aumenta la frequenza di funzionamento quando la macchina viene alimentata mediante il trasformatore e la riduce quando essa funziona a batteria, il sistema LongRun agisce continuamente e varia contemporaneamente alla frequenza anche la tensione di alimentazione della macchina. La potenza consumata da un processore in tecnologia CMOS dipende infatti da questi due parametri secondo la formula P=cv2f. Un apposito programma integrato nel firmware della macchina si occupa di tenere in ogni momento sotto controllo il carico della CPU e di adeguare in tempo reale (centinaia di volte in un secondo) i parametri di funzionamento. Grazie a questi accorgimenti il nuovo chip consuma molta meno energia rispetto alla concorrenza anche durante l'utilizzo delle normali applicazioni e ha un consumo praticamente nullo durante la fase di stand-by. In questo modo è possibile realizzare computer portatili con una maggior durata delle batterie e che non richiedono ventole di raffreddamento (a tutto vantaggio della silenziosità).

Transmeta cambia le regole del gioco

Il target dichiarato del nuovo processore è il mercato del "Mobile Internet Computing", ovvero dei palmtop computer, dei telefoni cellulari Web-enabled ed in generale dei pad computer, tutte applicazioni che necessitano di compatibilità col mondo x86 e di un consumo molto limitato di energia. E' in questo tipo di applicazioni infatti che la tecnologia LongRun offre i maggiori vantaggi. In un notebook infatti la CPU consuma solamente un quarto dell'energia totale del sistema. Il rimanente viene utilizzato per le diverse periferiche presenti sulla macchina, in particolare per l'hard disk, la ventola di raffreddamento e la retroilluminazione del display.

Per questo motivo in questo tipo di applicazioni ci si può aspettare al massimo un risparmio di circa il 30% nei confronti di una CPU tradizionale. In un palmtop o in un Web pad invece il risparmio di energia rispetto ad una CPU tradizionale diventa sensibile. E' pur vero che esistono delle CPU che consumano ancora meno, ad esempio la piattaforma StrongArm, tuttavia esse non sono compatibili x86 e perciò non sono in grado di utilizzare direttamente Windows. Il poter utilizzare applicazioni scritte per la versione x86 di Linux è un fattore non trascurabile, in quanto in un Web pad rende possibile l'utilizzo diretto di molti plug-in per il browser che altrimenti non sarebbero disponibili.

Un ulteriore vantaggio offerto dalla compatibilità con la piattaforma x86 è la disponibilità immediata di tool di sviluppo e di applicazioni pronte a funzionare senza modifiche sul nuovo hardware.

L'utilizzo della tecnologia CMS permette, senza dover riprogettare l'hardware ma riscrivendo solamente del software, di far girare sulla stessa macchina applicazioni scritte per qualunque altra piattaforma, passata o futura. Il tutto può essere implementato da Transmeta in tempi ristretti e senza dover dipendere da altre aziende. Il controllo sia dell'hardware che del software pone Transmeta in diretta concorrenza con Intel e con gli altri fabbricanti di software (e infatti questa è la politica dell'azienda nei confronti del mercato OEM). I chip vengono fabbricati in collaborazione con IBM e venduti direttamente al mercato OEM da Transmeta.

Transmeta ha la possibilità di correggere eventuali bug nell'hardware semplicemente passando al team di ingegneri che lavora alla nuova versione del morpher il vincolo di non utilizzare una determinata sequenza di istruzioni.

Per applicare la patch non sarà più necessario richiamare tutte le macchine (come insegna il caso del bug nella divisione del Pentium), ma tale operazione potrà essere effettuata direttamente dall'utente finale senza dover aprire la macchina. Infine è notevolmente più semplice aggiungere nuove caratteristiche lavorando sul software piuttosto che riprogettando l'hardware. Le nuove funzionalità possono poi essere spedite agli OEM semplicemente mediante la posta elettronica.

Il ruolo di Linus Torvalds

Transmeta, allo scopo di offrire al mercato OEM una soluzione hardware-software completa, ha realizzato "Mobile Linux", una versione del sistema operativo appositamente pensata per funzionare in sistemi senza hard disk, quali mobile Internet devices, palmtop, Web pad e telefoni cellulari. Mobile Linux verrà prossimamente distribuito secondo la licenza GPL. Il ruolo di Linus Torvalds come ingegnere della casa californiana è stato quello di collaborare allo sviluppo di Mobile Linux e del Code Morpher.