PERL: Lavorare con i FILE

Tomaso Minelli
Dipartimento di Scienze Statistiche
Università di Padova

Operatori su file
Lettura e scrittura di file

PERL è un linguaggio che nasce per operazioni sistemistiche e per questo motivo dispone di un ampio set di costrutti per semplificare il lavoro sul filesystem.
Esistono allo scopo una serie di operatori che solitamente ritornano una variabile booleana (true o false) per accedere direttamente alle informazioni sui file, ad esempio il codice

```
$file='/etc/passwd';
if (-e $file) {
    print "Il file passwd esiste";
}
```

sfrutta l'operatore `-e` che ritorna true nel caso il file esista, false altrimenti.

Esistono anche altri operatori a ritorno booleano per i file:

-f true se è un file
-d true se è una directory
-l true se è un link
-T true se è un file di testo
-B true se è un file binario
-z true se il file non ha dimensioni nulle (zero-size)
-r true se il file è leggibile
(dall'utente che esegue lo script)
-w true se il file è scrivibile (writable)
-x true se il file è eseguibile (eXecutable)

in ambiente unix esistono file di testo che sono anche eseguibili, come ad esempio gli script PERL.

Esistono altri operatori che ritornano delle informazioni specifiche sul file:

-s ritorna la dimensione (size) in byte

-M ritorna il tempo passato dall'ultima modifica

-A ritorna il tempo passato dall'ultimo accesso

```
print “il file ha dimesione “.
    (-f $file).”\n”;
```

```
#!/usr/bin/perl
print “Inserisci il nome di un
file”;$file=<STDIN>;
chomp($file); #elimina il carattere \n
if -r $file print “Il file è leggibile\n”;
if -d $file print “Il file è una directory\n”;
if -T $file print “E' un file di testo\n”;
```

Per aprire un file per analizzarlo, crearlo o manipolarlo con il nostro script usaimo la funzione open:

```
open FILE_HANDLE, $MODO, $NOME_FILE;
```

il parametro FILE_HANDLE è il parameto che utilizzeremo per accedere al file, ad esempio:

```
print FILE_HANDLE
    "del testo da inserire";
$riga=<FILE_HANDLE>;
@righe=<FILE_HANDLE>;
read(FILE_HANDLE,
    $buffer,$numero_byte);
```

fare attenzione al fatto che FILE_HANDLE non necessita del carattere `$`, poiché NON si tratta di una variabile scalare, ma propriamente di una "puntatore" al file.

Il modo di apertura del file è una stringa:

modo	lettura	scrittura	append	creazione	cancellazione
<	x				
>		x		x	x
>>		x	x	x	
+>	x	x			
+>>	x	x	x	x	x
| COM		x			
COM |	x				

gli ultimi due modi prevedono la possibilità di aprire uno stream attraverso un pipe (|) con un comando (COM), ad esempio

```
open STREAM, "ls -la /etc |"
```

ci permetterà di analizzare l'output di ls -la /etc

Il comando open può essere anche sintetizzato (generalmente) unendo il modo di apertura al file:

```
open FILE, "</etc/passwd";
```

se non viene specificato nessun modo di apertura si intende sottinteso che si voglia aprire in sola lettura (<).

Per avere informazioni su un file si può utilizzare la funzione stat che ritorna un array contenente tutte le informazioni che lo caratterizzano (alcune delle quali sarebbero ricavabili con gli operatori descritti in precedenza):

```
@file_info=stat(FILE_HANDLE);
$dimensione=$info[7];
    #dimensione in byte del file
read(FILE_HANDLE,
    $buffer, $dimensione);
    #salva l'intero file nella
    #variabile buffer
```

Gli altri elementi ritornati dalla funzione buffer sono:

0	Numero identificativo del filesystem
1	Numero identificativo dell'inode
2	tipo e permessi del file
3	numero di link al file
4	UID del proprietario
5	GID del proprietario
6	Numero identificativo del device (file /dev/)
7	dimensione del file (in byte)
8	timestamp ultimo accesso
9	timestamp ultima modifica
10	timestamp ultimo cambiamento di inode
11	dimensione blocco consigliato per I/O
12	numero dei blocchi allocati

Abbiamo visto come aprire un file con la funzione open, come leggerlo con la funzione read oppure utilizzando lo stream, ora vediamo come scriverlo: usiamo la stessa funzione print di stampa a video:

```
print FILE_HANDLE "testo";
```

poiché quando noi scriviamo print, perl considera sottointeso l'handle a STDOUT.
Possiamo poi sapere in che posizione (su quale byte) del file ci troviamo utilizzando la funzione `tell`, o spostaci in esso utilizzando la funzione `seek`:

```
$posizione=tell(FILE_HANDLE);
```

`$posizione` sarà il numero di byte dall'inizio del file. L'inizio del file è un valore pari a 0.

Per spostarci nel file usiamo la funzione seek:

```
seek(FILE_HANDLE,
    $numero_di_byte, $da_dove);
```

il secondo parametro ci indica di quanti byte (o caratteri) spostarci, il terzo indica da dove contare i bye dello spostamento:

valore	calcola da
0	calcolati dall'inizio del file
1	calcolati dalla posizione corrente
2	calcolati dalla fine del file

Lo stream STDIN è utile anche per interagire con l'utente:

```
print
    “Scrivi 1, 2 o altro e premi in-
vio...”;
$valore=<STDIN>;
chop(valore); #elimina il car. \n
if ($valore==1) {
    print “Hai scritto uno\n”;
} elsif ($valore==2) {
    print “Hasi scritto due\n”;
} else {
    print “Hai scritto qualcosa diver-
so da uno o due\n”;
}
```

Invece di usare l'istruzione open con modalità di apertura |, si può recuperare il risultato di un comando esterno in altre due maniere, usando gli apici inversi oppure l'operatore `qx`:

```
$lista=`ls -la /etc/`;
```

equivalentemente

```
$lista=qx/la -la \/etc\/;
```

ricordasi sempre l'escape sui caratteri non alfanumerici.

Può servire eseguire un comando senza bisogno di conoscere l'output, ma esclusivamente il valore di ritorno; in questo caso usiamo la funzione system:

```
$status = system("ls", "-la");
```

il cui primo parametro è il comando, il secondo sono le eventuali opzioni.

Si può usare anche la funzione `exec`, che ha la stessa sintassi di `system`, ma provoca la terminazione del nostro script, ovvero equivale a `system(...); end;`

E' il caso di ricordare che la variabile `$?` ci fornisce il valore di ritorno dell'ultima operazione esterna (pagina 11 lucidi 02)