

# 01 Presentazione del linguaggio PERL

## 1.1 Genesi ed Evoluzione di un Linguaggio

Quando si parla di linguaggi di programmazione vengono alla mente i soliti noti: C, Basic e Pascal dai quali ne derivano una fitta serie. Nella realtà però non si pensa più al linguaggio di per sé ma al Framework della Software House che lo ha commercializzato, per cui parliamo di Visual Basic e Visual Studio di Microsoft, di Delphi e C Builder di Borland (Java fa un po' storia a sé). Si ragiona in termini di compilatori, ovvero di programmi che generano programmi. Molto spesso però ci si trova ad affrontare problemi contingenti che necessitano di soluzioni rapide e magari cross platform, ovvero di piccole routine che possano girare su più piattaforme. Certo, il C presenta soluzioni multiplatforma col celeberrimo GCC (Gnu C Compiler) che è aperto a qualsiasi architettura di sistema. Stiamo però parlando di un linguaggio di programmazione molto potente ma anche un po' ostico per i neofiti.

Il Perl, acronimo di *Practical Extraction and Reporting Language*, nasce nel lontano 1987 (la Versione 1.0 è del 18/10/87) per opera di Larry Wall (nella foto al termine di una conferenza tenutasi a Ferrara nel 2004) come un modo semplice ed immediato per l'elaborazione dei file di Configurazione e di Log (file in cui vengono registrate le attività compiute da un programma) che imperversavano (e lo fanno tutt'ora) nei sistemi Unix. Ha una sintassi che somiglia ad un C facilitato, a cui sono state eliminate tutte quelle particolarità che lo rendono di difficile comprensione. Ma la cosa che lo ha reso popolare è che, in un certo modo, poteva sostituire i diversi linguaggi di scripting (Korn shell, C shell e Bourne shell) che ogni distribuzione utilizzava. In particolare, il Perl permetteva di fare cose che normalmente venivano fatte combinando la semplicità del linguaggio delle shell (la famosa programmazione batch) con la potenza del linguaggio C.



## 1.2 Dieci Buoni Motivi per Usarlo

---

1. E' **multiplatforma**; a parte il tenere in considerazione le specifiche di ogni sistema operativo (ma per questo esistono delle funzionalità già integrate) come si usa dire per Java "write once, run everywhere", ovvero lo scrivi una volta e gira dappertutto.
2. **Facile da comprendere**; la sua curva di apprendimento consente al neofita di essere immediatamente produttivo ed al programmatore più esperto di scrivere codice molto complesso
3. **Tantissimi moduli** in circolazione (OpenSource) per qualsiasi esigenza.
4. Comprende un'**ampia comunità per il supporto**.
5. Trattandosi di un **linguaggio di scripting** libera il programmatore dalla gestione delle risorse della macchina su cui deve girare (memoria).
6. E' il **coltellino svizzero** per gli amministratori di sistema per tutte le problematiche di gestione che può risolvere.
7. E' **GRATIS!!!** Chiunque può scaricarselo ed utilizzarlo senza dover pagare alcunché.
8. E' **uno dei linguaggi più utilizzati** dalla comunità mondiale; controllando tutti i sorgenti presenti su sourceforge.net (il repository con oltre ottantamila progetti opensource) segue a ruota "mostri sacri" come C, C++, Java e PHP
9. E' un **linguaggio flessibile e robusto** che si adatta a qualsiasi esigenza dalla più semplice ed immediata alla più complessa ed articolata.
10. Se cominciate ad usarlo non lo lasciate più; ok, io sono un po' di parte, ma credo che alla fine di questo corso la maggiorparte di voi mi darà ragione. Come dice spesso Larry Wall: **"Il programmatore Perl si riconosce dal sorriso che porta stampato sulle labbra"**

## 1.3 Quali Piattaforme lo supportano?

---

Sono quasi certo di non essere smentito se dico che ogni piattaforma (c'è pure per l'Amiga) che meriti di esistere, possiede un porting per il Perl.

Le più importanti sono:

- Windows: 95/98/Me/NT/2000/XP/2003
- Unix : AIX/Solaris/HP-UX/
- Linux: tutte le distribuzioni
- Macintosh: Os/OsX

Se avete dei dubbi sulla vostra piattaforma preferita vi consiglio di dare un'occhiata al sito [www.cpan.org/ports](http://www.cpan.org/ports)

## 1.4 Requisiti

---

Non esistono dei particolari requisiti Hardware per far girare uno script in Perl, infatti se il vostro sistema operativo (che sia Windows o Linux) è abbastanza stabile da farci girare le normali applicazioni sicuramente lo sarà anche per l'interprete del linguaggio.

Per quello che riguarda la programmazione invece, avere esperienze di altri linguaggi procedurali come il Basic, Pascal o C può facilitare nella comprensione di termini come variabili, strutture di controllo, gestione input/output ec...

## 1.5 Lo Stile Perl

---

La filosofia Perl è "There's More Than One Way To Do It" (TMTOWTDI) - la stessa cosa può essere fatta in modi differenti - e questo si riflette sia nella sintassi che nell'uso dei comandi stessi. Per il Codice presente in questo Corso ho cercato, per quanto possibile, di mantenere lo stesso stile e coerenza. Sappiate però che, là fuori, è pieno zeppo di listati Perl scritti nella maniera più bizzarra e contorta mai concepita. In programmazione il concetto machiavellico de "Il fine giustifica i mezzi" è valido solo se poi chi mantiene il codice è la medesima persona che lo ha concepito, altrimenti è caos puro.

## 1.6 Le Caratteristiche del Linguaggio

---

- Linguaggio interpretato
- Case Sensitive (è sensibile al passaggio tra maiuscolo e minuscolo)
  - dei Comandi; sono sempre in minuscolo
  - delle Variabili; a seconda di come vengono inizializzate
- Object Oriented; ma in queste Lezioni verrà visto solo marginalmente per l'utilizzo dei Moduli esterni
- Essendo un interprete non produce un eseguibile in linguaggio macchina; lo fanno alcuni tools di terze parti ma attraverso un meccanismo di conversione dello script in linguaggio C per poi essere compilato
- Minimo foot-print; non occupa tanto spazio su disco

## 1.7 Siete pronti?

---

Nel corso di queste 30 Lezioni vedremo tutte le specifiche del linguaggio (variabili, statements, strutture di controllo, moduli, ecc...) e le applicheremo ad esempi concreti. Il Corso è orientato soprattutto alla scrittura di Scripting a linea di comando poiché è nella natura del linguaggio stesso; esistono però delle librerie che permettono di costruire delle vere e proprie interfacce grafiche (le famose GUI, Graphic User Interface) in stile Windows (anche per Linux) ma per la vastità degli argomenti da affrontare ci discosteremmo troppo dal Corso Base che queste Lezioni voglio intraprendere.

# 02

# Installazione e Configurazione

## 2.1 Quale Versione?

---

Da questo punto in avanti quando parliamo di Perl ci riferiamo alla **versione 5.x**; attualmente la 5.8 è quella considerata stabile ed è su quella che ho provato tutti gli script di questo Corso. Ormai da tempo si parla della versione 6 ma si tratta di una beta non ancora stabile per cui non verrà tenuta in considerazione.

## 2.2 Linux

---

Perl nasce in ambiente Unix per cui la maggioranza delle distribuzioni sul mercato (acquistabili o scaricabili liberamente dalla rete) ne prevedono già una versione installata, per averne la riprova aprendo una shell di terminale (konsole o xterm a scelta) e digitando

```
whereis perl
```

Si dovrebbe ottenere la lista dei percorsi (contenuti nella variabile PATH) che contengono perl; solitamente si ha questo risultato:

```
perl: /usr/bin/perl /usr/X11R6/bin/perl
```

quello che ci interessa è il primo percorso, ovvero /usr/bin/perl che useremo come primo riferimento per la prima riga presente su ogni sorgente.

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

questa prima riga dice al sistema dove andare a reperire l'interprete dei comandi.

Se così non fosse occorrerà installarla ex-novo; sicuramente in qualche CDROM della vostra distribuzione preferita (Suse, Mandrake, Fedora, Debian, Gentoo) è presente nel formato più adatto (.rpm per Suse, Mandrake e Fedora, .deb per

debian ed .ebuild per Gentoo). Al limite, andando sempre sul sito dell'ActiveState (vedi nel paragrafo per Windows) si potrà scaricare e scompattare (`tar -xvf ActivePerl-5xxxx-i686-linux.tar.gz`) il pacchetto per Linux dopodichè da file di setup lo potete installare.

### 2.3 Windows

Sotto Windows esiste un pacchetto distribuito dalla Active State, una Software House specializzata nella produzione di tools per programmatori, che ve ne consentirà il libero download al seguente url:

[www.activestate.com/Products/Download/Download.plex?id=ActivePerl](http://www.activestate.com/Products/Download/Download.plex?id=ActivePerl)

Ciccando su MSI (è il formato che usa Windows per i propri pacchetti software) il gioco è fatto.

*Gli indirizzi internet rappresentano una delle cose più variabili del mondo, per cui se si dovesse ricevere una pagina non trovata basterà risalire al dominio del sito [www.activestate.com](http://www.activestate.com) e cercarsi il link di download per Windows.*

Una volta scaricati, gli oltre 12 Mb andranno mandati in esecuzione come un qualsiasi altro pacchetto di windows col suo bel setup assistito.



E' importante fare un'**installazione completa** con PPM 3.0 ed Examples inclusi, al termine della quale troverete il vostro bel interprete PERL.EXE sotto

C:\PERL\BIN (sempre che in fase d'installazione non abbiate modificato qualcosa)

Sotto Windows a differenza di Linux non è importante che la prima riga di un programma Perl contenga lo She-Bang (#!) perché è il registry a dire con quale programma mandare in esecuzione un file, inoltre usando il comando:

```
c:\perl\bin\perl.exe prova_perl.pl
```

si esegue lo script direttamente da prompt MsDos

Usando lo she-bang alla Unix siete sicuri che lo stesso script vi funzionerà sia sotto Linux che sotto Win32, se usate quella col percorso di Windows invece dovrete cambiarla ad ogni passaggio di Sistema Operativo.

## 2.4 Sorgenti .pl

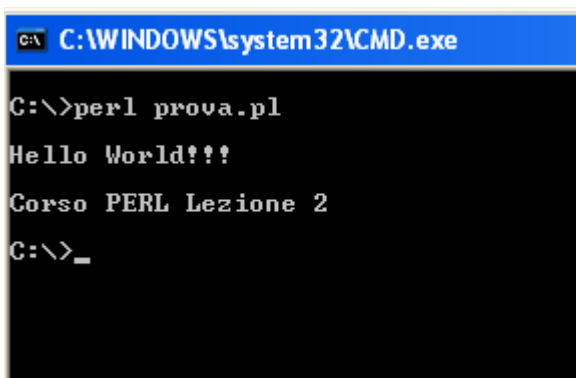
I sorgenti sono dei file di testo (o Ascii per quelli che parlano bene) che contengono le righe di codice per l'interprete Perl.

Usando un qualsiasi editor di testo e scrivendo il seguente codice:

```
#!/usr/bin/perl

print "\nHello World!!!\n";
print "\nCorso PERL Lezione 2\n";
```

salvandolo con l'estensione .pl e lanciandolo col comando `perl prova.pl` otterrete l'esecuzione del vostro primo script Perl



Per inserire dei commenti in un sorgente, ovvero quelle parti di testo che servono a spiegare il funzionamento dello script e non devono essere eseguite dall'interprete dei comandi, si usa il simbolo #

Quando trovate tante righe che cominciano con questo simbolo significa che siete fortunati perché l'autore del codice è un tipo molto comunicativo e vuole spiegarvi per filo e per segno il funzionamento del suo programma in modo che anche un estraneo possa apporvi delle modifiche a proprio uso e consumo.

```
#!/usr/bin/perl
#
# Autore: Alessandro Carichini
# Data: 29-05-2005
# Note: Esempio di prova per far comprendere le funzionalità
#       di uno script in perl
#
#
```

```
print "\nCiao Mondo";
```

```
# Questa è una riga commentata che l'interprete Perl non esegue
```

Una nozione importante è quella delle istruzioni Case Sensitive, il Perl è infatti sensibile al fatto che tutte le istruzioni siano in minuscolo.

**Se siete dei programmatori a cui piace scrivere le istruzioni o completamente maiuscole oppure minuscole ma con la prima lettera maiuscola dovete scordarvelo, il Perl non ammette free-styling nel nome dei propri comandi.**

Per cui se nel prosieguo del Corso doveste accorgervi di questa mancanza, consideratelo un refuso (maledetto correttore ortografico di Word!!!). Abituatevi perciò fin da subito a scrivere tutto in minuscolo (tranne il nome delle variabili, in quel caso potete fare come volete).

# 03

## Ambiente PERL & C.

### 3.1 Un ambiente spartano

---

Come ho ripetuto fino allo sfinimento, il Perl nasce sotto Unix per cui il suo ambiente di sviluppo è piuttosto spartano; un editor di testo (magari il `vi` o l'`emacs`) l'interprete, la shell in linea di comando ed il gioco è fatto.

Anche sotto Windows il discorso non cambia se non fosse che ci vengono in soccorso tanti tools (Open Source se possiamo scegliere) che forse non ci forniscono un framework vero e proprio ma ci permettono di sviluppare codice in modo più confortevole.

### 3.2 Editor per Windows

---

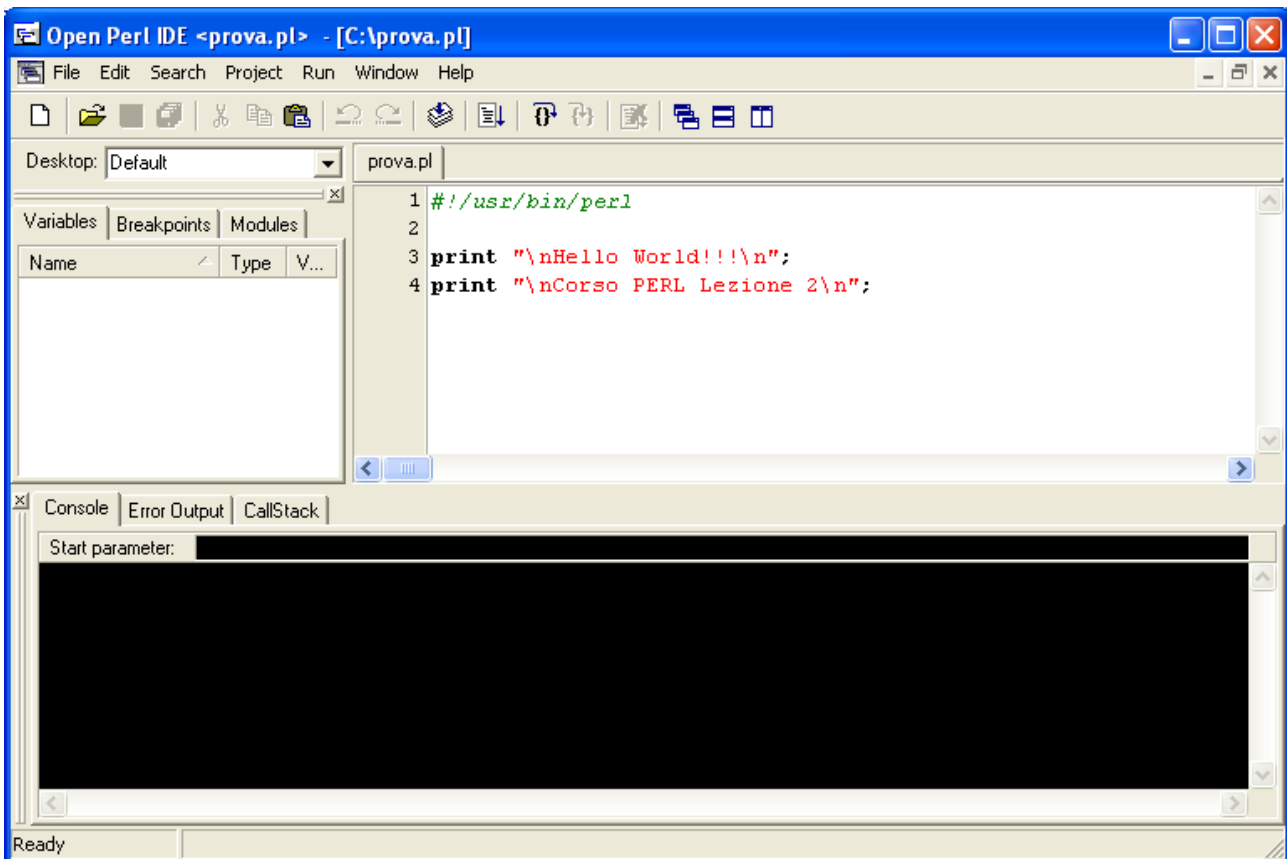
Per Windows esistono una miriade di editor di testo, dal più semplice ed integrato Notepad (che però vi forza sempre l'estensione a `.TXT`) al completo ma commerciale UltraEdit32.

Sicuramente ne esisteranno di altrettanto validi, ma per esperienza personale mi sono trovato molto bene con questi due prodotti completamente free:

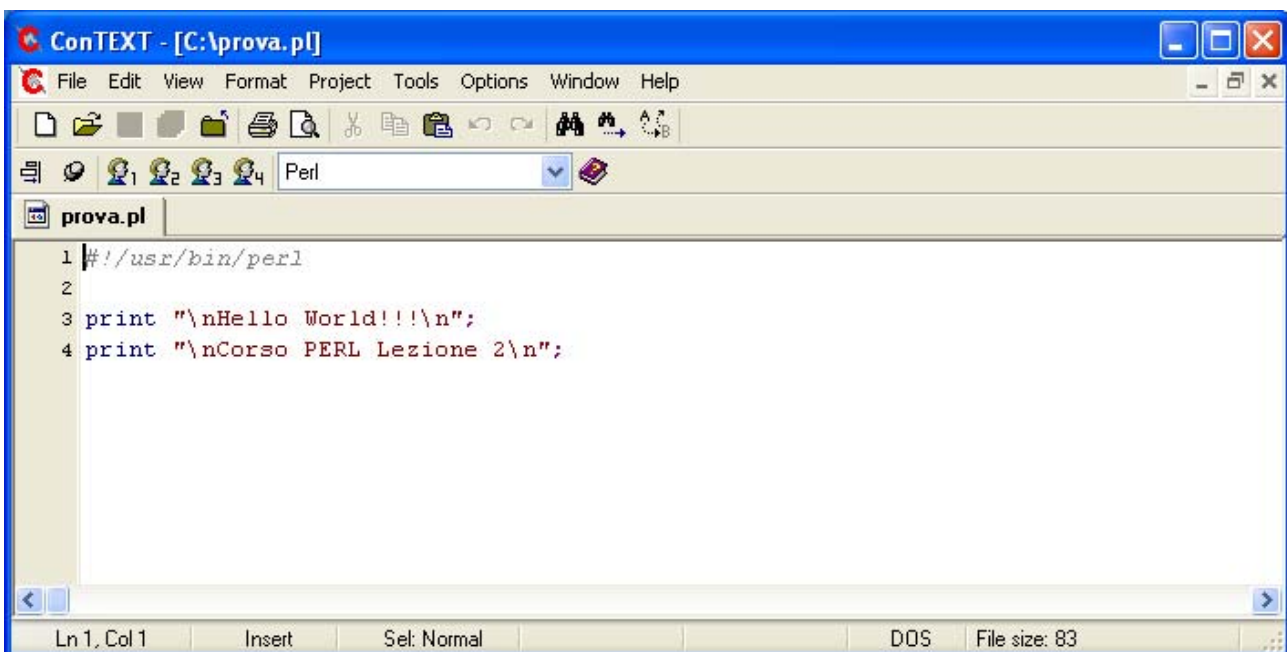
- Open Perl IDE [open-perl-ide.sourceforge.net](http://open-perl-ide.sourceforge.net)
- ConTEXT [www.context.cx](http://www.context.cx)

**Open Perl IDE** è un ambiente integrato di sviluppo per scrivere e fare il debugging (vedi Capitolo 11) di script in Perl sotto Windows. E' scritto in Delphi 5 - per cui è possibile pure modificarsi l'editor a proprio piacimento - ed è quello che più si avvicina all'idea di framework, anche se gli manca un elemento importante come l'auto-completamento delle istruzioni digitate e un help contestuale integrato (ma forse sono features che verranno implementate in futuro).





**ConTEXT** prevede un'interfaccia simile ai tanti Editor di file di Testo presenti sul mercato, ha la funzionalità di Syntax Highlighting (evidenziazione della sintassi) che non riconosce solo il Perl ma anche tanti altri linguaggi ed è molto leggero.



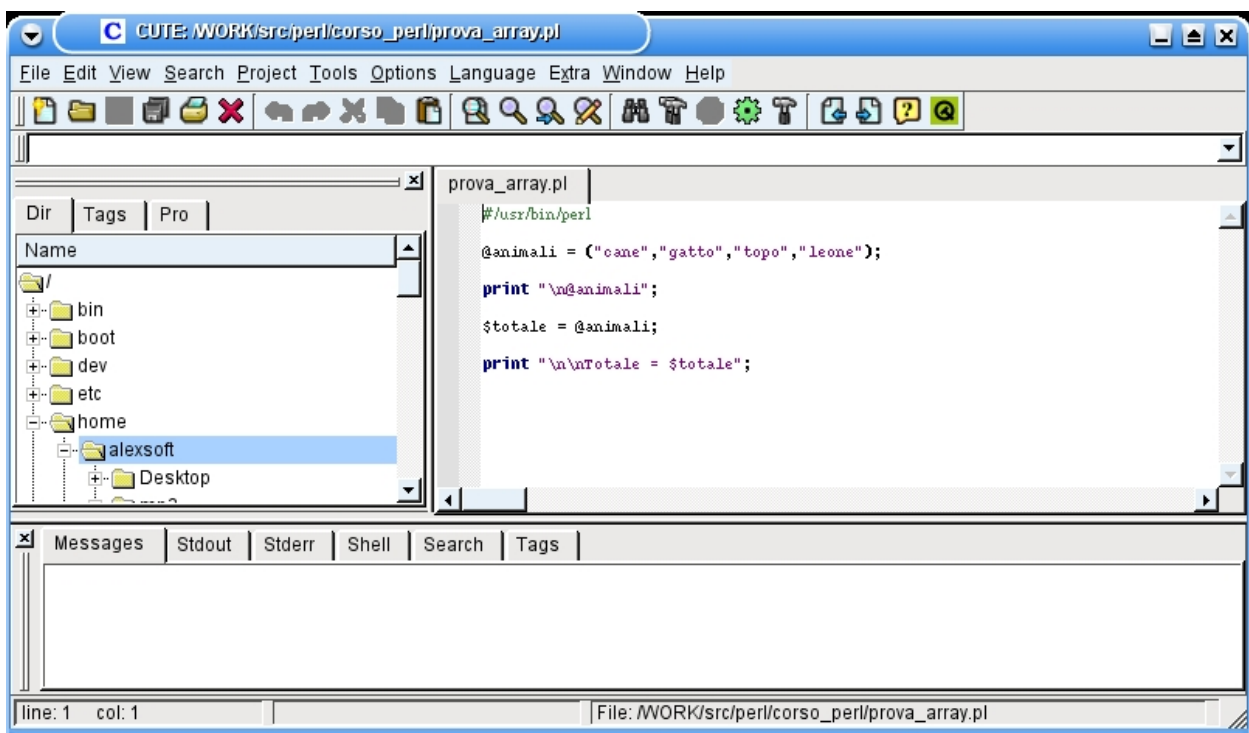
Chiaramente PerlIDE è un ambiente più completo rispetto ad un semplice Editor di testi come ConTEXT. Il mio consiglio è quello di scaricarseli entrambi e vederne le potenzialità per poi scegliere il meglio.

### 3.3 Editor per Linux

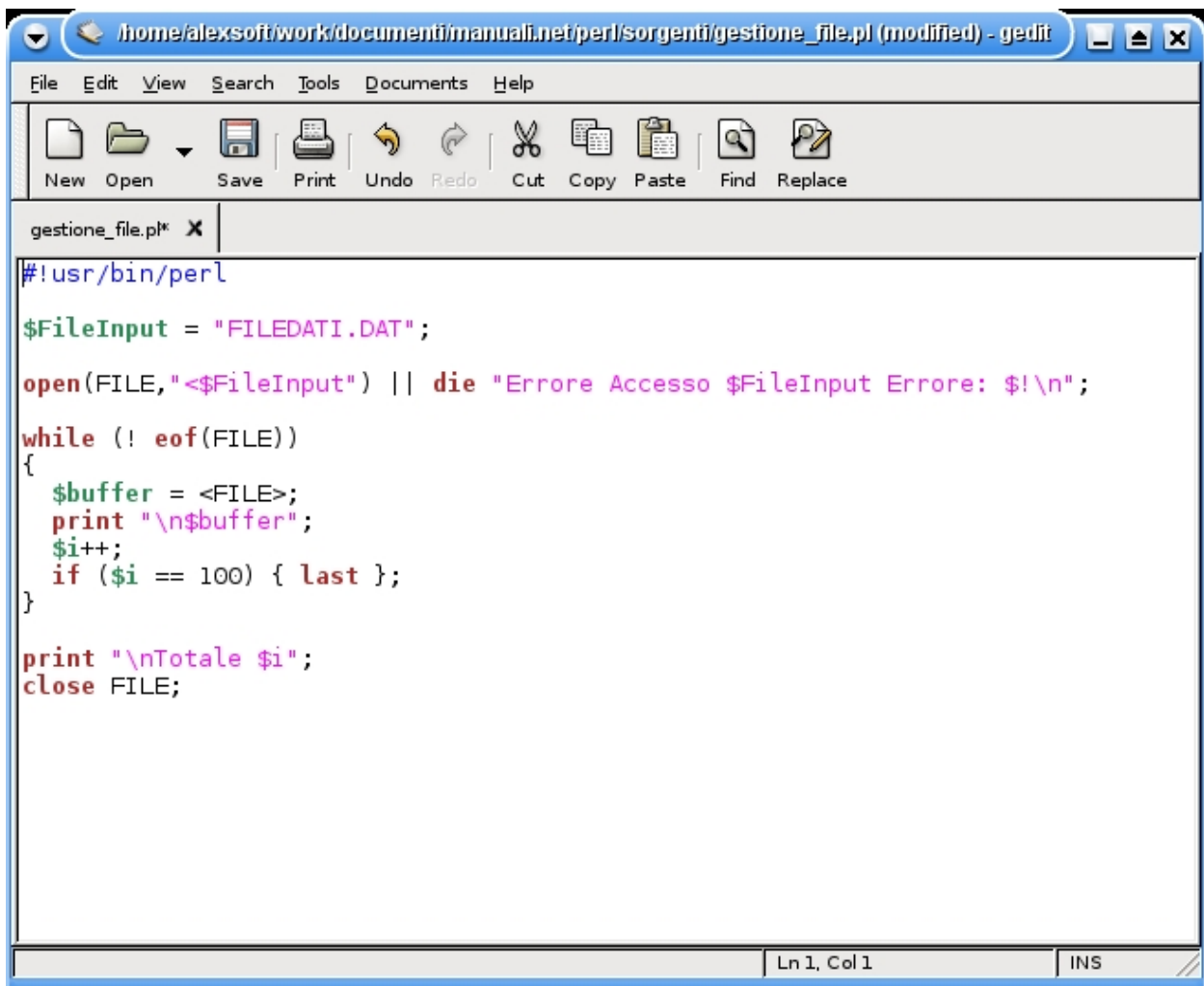
Anche Linux prevede tanti editor grafici simili a quelli per Windows, i più significativi sono:

- CUTE [cute.sourceforge.net](http://cute.sourceforge.net)
- GEDIT [www.gnome.org/projects/gedit](http://www.gnome.org/projects/gedit)

**CUTE** è un Editor di sorgenti (utilizza le librerie QT su cui si poggia il KDE) che riconosce parecchi formati, tra cui anche il Perl. A mio parere è uno dei migliori ma che però, come tutti gli editor non dedicati, non offre funzionalità specifiche come l'auto-completamento delle istruzioni o il debug del codice.



**GEDIT** è simile al CUTE con l'unica differenza che è un editor già integrato in qualsiasi distribuzione Linux (basta includere le librerie di Gnome) anche se non è specifico per il Perl pur riconoscendone la sintassi (come pure quella di tanti altri linguaggi).



```
#!/usr/bin/perl
$FileInput = "FILEDATI.DAT";
open(FILE,"<$FileInput") || die "Errore Accesso $FileInput Errore: $!\n";
while (! eof(FILE))
{
    $buffer = <FILE>;
    print "\n$buffer";
    $i++;
    if ($i == 100) { last };
}
print "\nTotale $i";
close FILE;
```

### 3.4 Tools Vari

Putroppo la stragrande quantità di tools di sviluppo per il Perl ha una natura commerciale, se provate a fare un salto sul sito [www.activestate.com](http://www.activestate.com) troverete dei veri e propri framework alla Visual Studio e affini. Tra i prodotti Open Source invece non c'è tutta questa abbondanza ma per programmare bene in Perl sono sufficienti i prodotti appena descritti.

### Domande di Fine Modulo

1. Che differenza c'è tra un linguaggio compilato ed uno interpretato?
2. Quali sono gli elementi necessari per far girare uno script in Perl?
3. Su quali piattaforme può essere eseguito?

# 04

## Strutture Dati: Le Variabili

### 4.1 Le Variabili Scalari

Le variabili sono dei contenitori di dati, in Perl non ne esiste una tipizzazione come in altri linguaggi tipo C o Basic (numero intero, stringa alfanumerica, vero/falso, numero doppia precisione, ecc..) la loro natura cambia a seconda di quello che contengono:

```
$alfa = 'CASA';
$beta = 5;
$gamma = 5.10;
```

in questo caso assegno ad \$alfa la stringa CASA, a \$beta il numero 5 e a \$gamma il numero frazionato 5,10 (o virgola mobile).

Ma se poi cambio tutto con:

```
$alfa = 5.10;
$beta = 'CASA';
$gamma = 5;
```

non c'è problema se non il rischio di impazzire in fase di Debugging (la fase di sistemazione degli errori di un programma che vedremo in seguito) per cui fate molta attenzione a cambiare la natura delle variabili all'interno dello stessa procedura.

**Il Perl chiama scalari le variabili semplici, ovvero quelle che possono contenere un unico elemento.**

#### 4.1.1 Caratteristiche di una Variabile Scalare:

- Simbolo **\$** davanti al nome
- Il nome può essere lungo al **massimo 256 caratteri**
- E' **Case Sensitive** per cui fa differenze tra maiuscolo e minuscolo, quindi la variabile \$casa è diversa da \$Casa
- Contiene un **singolo elemento**

### 4.1.2 Variabili Numeriche

Il Perl tratta tutti i valori numerici allo stesso modo che siano interi, decimali a singola o doppia precisione

Gli Operatori Numerici sono:

+	Addizione	-	Sottrazione
*	Prodotto	/	Divisione
%	Resto della divisione	** n	Potenza di n
++	Incremento	--	Decremento

```
$alfa = 5;
$beta = 10;
print ($alfa + $beta * 2)/100;
```

ma se vogliamo il semplice incremento di una variabile anziché scrivere

```
$alfa = $alfa+1;
```

useremo l'operatore ++, ottenendo:

```
$alfa++;
```

se lo vogliamo decrementare:

```
$alfa--;
```

### 4.1.3 Variabili Alfanumeriche o Stringhe

Le stringhe sono delle sequenze di caratteri (Ascii) racchiuse all'interno di apici singoli o doppi.

Se assegniamo a due variabili la medesima stringa e le mandiamo in output, avremo il medesimo risultato

```
$alfa = 'casa mia';
$beta = "casa mia";
print $alfa;
print $beta;
```

Gli Operatori Alfanumerici sono:

.	Concatenazione
X	Ripetizione
Substr ( \$stringa, \$daPos, \$nCaratteri )	Sottostringa
Index ( \$subStringa, \$stringa )	Posizione

Quindi se voglio concatenare due stringhe userò:

```
$gamma = $alfa . $beta;
```

Se invece voglio replicare un certo numero di volte una stringa userò la *x* (minuscola mi raccomando).

```
$riga = "-" x 80;
```

per estrarre una sottostringa da uno scalare; `substr()`

```
$codice = substr($elenco,0,4);
```

e per sapere da che posizione parte una sottostringa in uno scalare; `index()`

```
$nPosizione = index("CB01",$elenco);
```

ma queste funzioni, che sono alla base della manipolazione delle stringhe, le vedremo continuamente nei capitoli ed esempi successivi.

Le **Sequenze di Escape** sono dei caratteri speciali che non vengono stampati ma eseguono delle azioni. Vengono formate dal backslash (\) in aggiunta ad uno o più caratteri

Seq.Escape	Significato
\n	New Line (Sotto Win32 nei file di testo le righe terminano con \n e \r mentre sotto Unix solo con \n)
\r	Return (ritorno a capo)
\t	Tabulazione
\f	Form Feed (Salto pagina)
\b	Backspace (cancella ultimo carattere)
\e	Esc
\\	Backslash
\"	Doppio Apice
\l	Trasforma il carattere che segue in minuscolo (LOWER)
\L	Trasforma tutti i caratteri che seguono in minuscolo
\u	Trasforma il carattere che segue in maiuscolo (UPPER)
\U	Trasforma tutti i caratteri che seguono in maiuscolo
\v	Tabulazione verticale

Se voglio stampare il carattere di backslash lo devo inserire due volte per evitare che venga interpretato come una sequenza di escape. Un esempio lo vediamo quando lavoriamo sui filesystem di Windows:

```
$percorso = "C:\\WINDOWS\\SYSTEM32\\";
```

L'utilizzo del backslash è importante nel caso in cui si voglia usare il doppio apice (") come carattere anziché come chiusura/apertura di una stringa.

```
$stringa = "Jon \"Maddog\" Hall";
print $stringa;
```

Jon "Maddog" Hall

## 4.2 Le Virgolette

Quando si lavora con Perl (ma anche con la maggior parte dei linguaggi che provengono da Unix) ci si imbatte nell'utilizzo delle virgolette sia in ambito di variabili che di output. Ne esistono di tre tipologie:

Simbolo	Descrizione	Codice Ascii
'	Singolo Apice	39
"	Doppio Apice	34
`	BackQuotes	96

Il **Singolo Apice** (l'apostrofo) è il più semplice perché viene usato per stampare o assegnare una stringa così com'è.

```
$citta = 'Rimini';
```

Il **Doppio Apice** (le classiche virgolette) invece esegue quella che viene definita Interpolazione della stringa, più facile da vedere che da spiegare:

```
$casa = "Io Abito a $citta";
$casa = 'Io Abito a $citta';
```

nel primo caso mi sostituirà la variabile `$citta` con quello che gli ho precedentemente assegnato, nel secondo invece manterrà il contenuto indicato.

Il **Backquotes** (un apostrofo girato) invece manda in esecuzione il contenuto della stringa ed il risultato lo assegna alla variabile.

```
$SistemaOperativo = `echo %OS%`;
```

Assegniamo al nostro scalare il contenuto della Variabile d'Ambiente OS (presente da Windows NT in avanti).

### 4.3 Gli Array

Un Array è un insieme di elementi (o lista) avente un indice numerico come identificatore. Per identificare un Array si usa il simbolo @

#### 4.3.1 Array Statico

```
@animali = ("cane", "gatto", "topo", "leone");
```

per identificare i singoli elementi posso usare la notazione scalare e l'indice di appartenenza, per cui

```
print $animali[0]; # cane
print $animali[1]; # gatto
```

se voglio usare un array per contenere degli intervalli di numeri o caratteri posso usare la seguente assegnazione

```
@numeri = (1..20);          # un array numerico riempito dall'1 al 20
@lettere = ("a" .. "z")    # un array alfanumerico dalla "a" alla "z"
```

#### 4.3.2 Array Dinamico

Posso creare un array con un solo elemento

```
@animali = ("cane");
```

e poi riempirlo con l'istruzione

```
push(@animali, "gatto");
```

Ma queste funzioni le vedremo di seguito in maniera più approfondita.

Un altro esempio:

```
@animali = ("cane", "gatto", "topo", "leone");
print "\nArray = @animali";
$total = @animali;
print "\n\nTotale Elementi = $total";
```

Il risultato sarà quello di ottenere la stampa del contenuto dell'array col print e del totale dei suoi elementi mettendo lo stesso array all'interno dello scalare \$total

Le funzioni standard per la gestione degli Array:

<code>push(@array, \$element)</code>	Inserisci un elemento in fondo all'array
--------------------------------------	--



<code>pop(@array)</code>	Toglie l'ultimo elemento di un array
<code>shift(@array,\$element)</code>	Toglie il primo elemento dall'array
<code>unshift(@array)</code>	Aggiunge un elemento all'inizio dell'array
<code>delete \$array[\$index]</code>	Rimuove un elemento dell'array in base all'indice

```
@array = ("rock","pop","classic");
# Aggiungo l'elemento "folk" in coda all'array
push(@array,"folk");
# Aggiungo l'elemento "blues" all'inizio dell'array
unshift(@array,"blues");
print "\n@array";
```

```
blues rock pop classic folk
```

#### 4.4 Gli Hash

Un Hash è un Array Associativo, ovvero un insieme di elementi avente come indice un valore alfanumerico. Ha lo stesso comportamento di un array col vantaggio di poter puntare ad un valore mnemonico piuttosto che numerico.

Il simbolo di riferimento è il `%` e la sua assegnazione avviene in questo modo:

```
%array_hash = (elemento_chiave,elemento_valore);
%temperature = ('Rimini',29,'Milano',32,'Roma',30,'Napoli',31);
```

ma possiamo usare anche un metodo più leggibile come:

```
%temperature = (
    Rimini => 29,
    Milano => 32,
    Roma => 30,
    Napoli => 31);
```

oppure se dobbiamo fare un'assegnazione dinamica (magari all'interno di un ciclo) ci conviene usare:

```
$temperature{$elemento_chiave} = $elemento_valore;
```

La **Chiave** è sempre considerata un'espressione scalare **stringa**, quindi anche quando si dovessero usare dei numeri verrebbero interpretati come alfanumerici quindi attenzione agli ordinamenti.

Per accedere alla tabella basta puntare all'elemento alfanumerico

```
print $temperature{'Rimini'};
```

29

Le funzioni standard per la gestione degli Hash:

<code>keys(%hash)</code>	Estrae solo gli elementi chiave dell'hash
<code>each(%hash)</code>	Estrae la coppia Chiave e Valore dall'hash
<code>delete(\$elemento{chiave})</code>	Elimina la coppia Chiave e Valore dall'hash
<code>exists ESPRESSIONE</code>	Verifica l'esistenza di una chiave nell'hash

```
# Estrazione delle sole chiavi da un hash ed inserite in un array
@citta = keys(%temperature);
```

### 4.5 I Reference

Si tratta di scalari che contengono l'indirizzo di memoria di un'altra variabile (scalare, array o hash). Sono una sorta di puntatore alla cella di memoria che la variabile utilizza per memorizzare i propri dati.

```
$variabile = 'casa';
$ref_variabile = \$variabile;
```

usando il **backslash** come simbolo con il backslash ed il nome della variabile noi andiamo ad inserire in uno scalare l'indirizzo di memoria della variabile stessa

```
print $ref_variabile;
```

Una volta che abbiamo il riferimento alla variabile abbiamo anche il suo contenuto, col simbolo \$

```
print $$ref_variabile;
```

**Nel perl non esiste il concetto di Variabile Costante**, ma questo può essere ovviato con l'utilizzo dei reference, infatti:

```
$pi_greco = \3.14;
$area = 2 * $$pi_greco;
```

Se poi volessi provare ad assegnare un nuovo valore a \$\$pi\_greco riceverei un messaggio di errore

Per un utilizzo quotidiano dei Reference vi rimando ai prossimi capitoli.

## 4.6 Le Variabili Speciali

Si tratta di variabili a cui il Perl ha dato un significato predefinito

### 4.6.1 Scalari Globali

La variabile speciale più comunemente usata è `$_`

Questa tabella riassume i più usati per ulteriori spiegazioni rimando ai prossimi capitoli in cui verranno trattati a seconda dell'argomento.

Simbolo	Nome Completo	Descrizione
<code>\$_</code>	<code>\$ARG</code>	Lo spazio di input e di ricerca del pattern predefinito
<code>\$.</code>	<code>\$NR</code>	Il numero di linea di input corrente relativa all'ultimo filehandle letto
<code>\$/</code>	<code>\$RS</code>	Il separatore di record di input, di default è l'avanzamento di linea
<code>\$,</code>	<code>\$OFS</code>	Il separatore di campo di output dell'operatore print
<code>\$\</code>	<code>\$ORS</code>	Il separatore di record di output dell'operatore print
<code>^\L</code>	<code>\$FORMAT_FORMFEED</code>	Il carattere per il FormFeed ( <code>\f</code> )
<code>\$0</code>	<code>\$PROGRAM_NAME</code>	Nome dello script in esecuzione
<code>\$!</code>	<code>\$ERRNO</code>	Descrizione dell'errore avvenuto
<code>^O</code>	<code>\$OSNAME</code>	Stringa indicante il Sistema Operativo in uso
<code>^X</code>	<code>\$EXECUTABLE_NAME</code>	Nome del file dell'eseguibile perl con il suo percorso assoluto

### 4.6.2 Array Globali

Simbolo	Descrizione
<code>@ARGV</code>	Array contenete gli argomenti della linea di comando
<code>@INC</code>	Array contenete la lista dei percorsi dove ricercare gli script in Perl
<code>@_</code>	Array di parametri per subroutine/funzioni

### 4.6.3 Hash Globali

Simbolo	Descrizione
<code>%INC</code>	Hash contenete il nome di ciascuno dei file che sono stati inclusi
<code>%ENV</code>	Hash contenente le variabili d'ambiente

## 4.7 Alcuni Esempi sull'uso delle Variabili Speciali

---

### **4.7.1 Passare i parametri dalla riga di comando**

Lanciando un semplice script da riga di comando:

```
# perl variabili_speciali.pl alfa beta gamma
```

Posso ottenere l'elenco dei parametri passati contenuti nell'Array @ARGV

```
foreach $parametro (@ARGV) {  
    print $parametro;  
}
```

Ed il nome dello script che ho lanciato con \$0

Questo può risultare molto utile nella creazione di utility parametrizzabili da riga di comando anziché da menu.

### **4.7.2 Leggere tutte le variabili di memoria**

Il Perl raccoglie le variabili di memoria del sistema operativo su cui viene ospitato (che sia Windows o Unix) in un hash avente come chiave il nome della variabile e come elemento il suo contenuto

```
$path_so=$ENV{ 'PATH' };
```

Vi mostrerà la variabile contenente tutti i percorsi che utilizza il vostro sistema operativo per la ricerca degli eseguibili

### **4.7.3 Quale Sistema Operativo?**

Può essere utile specialmente per script multi-piattaforma fare un controllo del sistema operativo in uso, in questo caso usando \$^O (\$OSNAME)

```
$sistema_op = $^O;  
  
if ($sistema_op eq "MSWin32") {  
    print "Microsoft Windows 9x/NT/XP/2000";  
}elsif ($sistema_op eq "linux") {  
    print "Unix/Linux";  
}elsif ($sistema_op eq "aix") {  
    print "Unix/Aix";  
}else {  
    print "Sconosciuto";  
}
```

## 4.8 Lo "Scope" delle Variabili

Il contesto di una variabile (scope) è il campo d'azione della variabile stessa; di default le variabili sono globali dal **namespace** da dove vengono inizializzate. Il **namespace** è un blocco di codice a cui abbiamo dato un nome, se non gli è stato dato espressamente è **main**.

Se invece vogliamo che il campo d'azione sia locale al suo namespace useremo **my** seguito dal nome della variabile. Nel caso di subroutine può essere comodo invece usare **local** per evitare conflitti con una variabile globale omonima.

```
my $casa,$casa2;
```

La cosa buffa è che **my genera variabili locali mentre local no**. Com'è possibile tutto ciò? In realtà **local** è rimasto solo per ragioni storiche di quando il Perl usava solo variabili globali (o variabili package) e c'era la necessità di un meccanismo che le salvaguardasse dall'uso omonimo all'interno di subroutine.

Comunque per farla breve è un bene seguire queste regole:

- Non usare le variabili globali
- Dichiarare sempre le variabili con **my**
- Non usare mai **local**

Nel Capitolo riguardante il Debugging approfondiremo ulteriormente questa norma di comportamento.

## 4.9 Riepilogando

In Perl esistono 3 tipologie di variabili:

\$	Scalare	Variabile semplice
@	Array	Variabile contenente più elementi indicizzati
%	Hash	Variabile contenente più elementi aventi una stringa come identificatore

Per cui scrivere

```
$citta
@citta
%citta
```

significa usare tre variabili diverse, se poi questi nomi vengono preceduti da un backslash, allora otterremmo l'indirizzo di memoria della variabile stessa, ovvero il suo Reference.

# 05 Strutture di Controllo

## 5.1 Le Condizioni (IF)

Ogni linguaggio che si rispetti prevede delle strutture per la verifica di determinate condizioni; il costrutto IF-ELSE permette di fare questo:

```
if ($a == 1) {
    print "Condizione 1";
}elsif ($a == 2) {
    print "Condizione 2";
}else {
    print "Condizione di Default";
}
```

Ogni blocco di condizione viene racchiuso da delle graffe, inoltre **anche se è possibile inserire degli IF-NIDIFICATI all'interno di blocchi ELSE è più corretto usare ELSIF** (non elseif come usa il PHP)

```
# NON CORRETTO
if ($a == 1) {
    print "Condizione 1";
}else {
    if ($a == 2) {
        print "Condizione 2";
    }
}
```

```
# CORRETTO
if ($a == 1) {
    print "Condizione 1";
}elsif ($a == 2) {
    print "Condizione 2";
}
```

Prestate attenzione anche al modo in cui si scrive il codice, facendo rientrare (indentare) le istruzioni che appartengono ad un blocco specifico e mettendo le parentesi graffe in questa maniera (Larry Wall docet).

## 5.2 Gli Operatori

### CONDIZIONALI

Comparazione	Operatore Numerico *	Operatore Stringa	
Uguaglianza	==	eq	Equal
Disuguaglianza	!=	ne	Not Equal
Minore di	<	lt	Less Than
Maggiore di	>	gt	Great Than
Minore o uguale	<=	le	Less Equal
Maggiore o uguale	>=	ge	Great Equal

\* Operatore Numerico significa che va usato in presenza di variabili numeriche mentre l'Operatore stringa per quelle alfanumeriche

### LOGICI

Operatore Logico	Simbolo	Descrizione
<b>AND</b>	&&	Vero quando sono vere entrambe le condizioni
OR		Vero quando è vera almeno una delle condizioni
NOT	!	Vero se l'espressione è falsa e viceversa

## 5.3 Le Iterazioni (Loop)

L'iterazione è il sale della programmazione, senza i cicli le operazioni ripetitive appesantirebbero inutilmente il codice ed il lavoro del programmatore.

Il Perl ne prevede di quattro tipi:

- While
- For
- Foreach
- Do / While

### 5.4 While

Il *While* è la classica iterazione arbitraria che ci permette di ripetere un blocco di codice finché la condizione (posta in cima) risulta soddisfatta.

```
while (<condizione>) {
    <azione>
}
```

```
$indice = 0;
while ($indice < 10) {
    print "Indice $indice";
    $indice++;
}
```

Questo loop ci stamperà 10 righe (da 0 a 9)

I **Modificatori di Ciclo** ci permettono di alterare la naturale sequenza di una iterazione, ne esistono tre:

- **Next**, ricomincia il ciclo dall'inizio
- **Last**, forza l'uscita dal ciclo
- **Redo**, ignora tutte le operazioni che lo seguono e salta alla prima istruzione del ciclo

```
$indice = 0;
while ($indice < 10) {
    print "\nIndice $indice";
    if ($indice == 5) {
        print "*** USO REDO ***";
        $indice++;
        redo;
    }

    print "*** OK ***";
    if ($indice == 8) {
        last;
    }
    $indice++;
}
```

In questo esempio l'uso del "Next" avrebbe causato un loop infinito

## 5.5 For

---

Il For è il ciclo per eccellenza, ossia quello predeterminato poiché già ne conosciamo la lunghezza. Un esempio classico è la stampa di un array.

```
for($indice = 0; $indice < 10; $i++) {
    print "\n$citta[$indice]";
}
```

## 5.6 Foreach

---

La comodità di questo operatore è il poter elencare una lista senza doverne conoscere la grandezza



```
@settimana = (lun,mar,mer,gio,ven,sab,dom);
foreach $week (@settimana) {
    print "\n$week";
}
```

Molto comodo quando l'iterazione riguarda un hash di cui non conosciamo la struttura

```
foreach $chiave (keys(%settimana)) {
    print"$chiave => $settimana{$chiave}\n";
}
```

### 5.7 Do / While

---

E' una variante del While che permette di eseguire il ciclo prima di verificare la condizione. Utile nei casi in cui la condizione muta all'interno del ciclo stesso.

```
$continua = 1;
do {
    If ($citta eq "MILANO") {
        $continua = 0;
    }
    $indice++;
} while ($continua);
```

I valori Booleani true e false nelle variabili Perl vengono sostituiti da 1 e 0 mentre nelle Condizioni si utilizza la terminologia di Vero e Falso.

### 5.8 In conclusione

---

Credo che l'utilizzo di diverse istruzioni per eseguire il medesimo ciclo di loop possa essere fuorviante, specialmente per chi inizia ad usare un nuovo linguaggio, per cui vi consiglio di usare solo questi tre a seconda delle occasioni:

- while per qualsiasi ciclo
- for per ciclare array statici
- foreach per ciclare array dinamici

Chiaramente è un puro consiglio, gli Statement in un linguaggio non vengono messi lì a caso e l'abbondanza a volte può dare la possibilità di alternare le situazioni a seconda della necessità, per cui questo non significa ignorarli del tutto ma focalizzare l'attenzione sui principali.

# 06

# Input/Output

## 6.1 Canali di Input/Output

---

Il Perl eredita la sua gestione dei canali di Input/Output dallo Unix (e quindi dal linguaggio C) per cui ne abbiamo tre predefiniti:

<b>STDIN</b>	Standard Input	Riceve in input tutto quello che viene digitato sulla tastiera
<b>STDOUT</b>	Standard Output	E quello che va in output sul video
<b>STDERR</b>	Output Errori	Output degli errori che viene gestito nei sistemi Unix

Ogni canale viene anche definito **Handle**

## 6.2 Output

---

Nella stragrande maggioranza dei linguaggi di programmazione la stampa in output viene gestita attraverso l'istruzione `print` ed anche il Perl non si sottrae a questa convenzione. Normalmente quando scriviamo

```
print "Stampa a Video\n";
```

Dovremmo pensare che è come se scrivessimo

```
print STDOUT "Stampa a Video\n";
```

infatti la funzione di `print` è quella di stampare su di un Handle aperto (in questo caso lo `STDOUT`) una determinata stringa.

Il `print` però permette di mandare in output una stringa semplice, se invece la voglio formattare in una certa maniera devo usare la funzione `printf` unita a dei simboli chiamati Specificatori di Campo nella forma:

%m.nx

m.n = larghezza del campo (il punto in caso di valori frazionati)  
 x = tipo di campo

Simbolo	Significato
%c	Carattere
%d	Intero con segno
%e	Virgola Mobile Esponenziale
%f	Floating, Doppia Precisione
%l	Long, Intero senza segno
%s	Stringa
%x	Esadecimale
%-	Allineamento a Sinistra
%%	Percentuale
%0	Usa lo zero anziché spazi per l'allineamento a destra

```
$importo = 6500.12;
printf("L'importo è di %10.2f", $importo);
```

L'utilizzo di più variabili avviene in modo semplice, prima con la formattazione all'interno dei doppi apici e poi con la sequenza dei nomi delle variabili separate dalla virgola

```
$importo1 = 6500.12;
$importo2 = 68.10;
$importo3 = 800.20;
printf("%10.2f %10.2f %10.2f", $importo1, $importo2, $importo3);
```

L'allineamento avviene a destra ma se, nel caso di stringhe, lo volessi fare a sinistra basterà aggiungere il – alla formattazione

```
$citta = "Bologna";
printf("Citta = %-30s", $citta);
```

### 6.3 Input

L'input dei dati avviene nel classico stile della redirectione in Unix ovvero attraverso l'uso del simbolo < seguito dal nome dell'Handle.

```
$input = <HANDLE>;
```

Nel caso di input da tastiera si utilizzerà l'handle STDIN

```
$input = <STDIN>;
```

l'interprete resterà in attesa della pressione del Return (o INVIO) che copierà assieme alla stringa digitata nello scalare \$input

La funzione **chomp** permetterà di eliminare quest'ultimo carattere e lasciare alla variabile il solo contenuto digitato.

```
chomp($input=<STDIN>);  
if ($input == 1) {  
    print "Scelta n.1";  
}elsif ($input == 0) {  
    print "Esci";  
}
```

A volte può essere comodo inserire in input da codice del testo formattato in una certa maniera (magari perché il codice rimane più leggibile) , inserendo l'operatore << associato ad una stringa mnemonica, possiamo dire ad una variabile di accettare tutto quello che segue finché non si scontra con un pattern arbitrario (deciso da noi) che gli intima lo stop.

```
$crea_table = <<MYSQL_PATTERN_STOP  
    CREATE TABLE ABICAB (  
        ABI        CHAR(5),  
        BANCA     CHAR(50),  
        CAB       CHAR(5),  
        COMUNE   CHAR(40),  
        CAP      CHAR(5),  
        PROV     CHAR(2),  
        LOCALITA CHAR(40),  
        SPORT    CHAR(40),  
        VIA      CHAR(40)  
    )  
MYSQL_PATTERN_STOP  
;
```

il pattern in questo caso è `MYSQL_PATTERN_STOP` che segnala la fine dell'input.

Per la gestione dei file vi rimando all'omonimo Capitolo.

# 07

## SubRoutine e Funzioni

### 7.1 Divide et Impera

I latini non usavano il computer per programmare, ma avevano chiaro il concetto che un problema complesso è più facile da gestire se viene scomposto in tanti piccoli problemi più semplici da risolvere. In Perl possiamo decidere di scrivere tutta un'applicazione su di un unico file e con poche funzioni oppure di segmentare il tutto in tanti files e moduli separati. Chiaramente è la seconda la giusta via da seguire specialmente quando si ha a che fare non con un semplice script di un ventina di righe ma con un'applicazione complessa.

Prende vita così sia il concetto di suddivisione logica (tanti sottoprogrammi o subroutine) che fisica (diversi files o moduli). Una Subroutine non è altro che una parte di codice raggruppata per svolgere una determinata azione. A volte questo avviene solo per motivi di una migliore leggibilità del codice, altre perché si tratta di un'operazione ripetitiva.

```
sub MenuPrincipale {
    print "\n[1] Inserimento";
    print "\n[2] Modifica";
    print "\n[3] Stampa";
    print "\n[0] Esci";
    $scegli = <STDIN>;
    if ($scegli == 1) {
        MenuInserimento;
    }elseif ($scegli == 2) {
        MenuModifica;
    }
}
```

Le Subroutine possono diventare anche delle vere e proprie Funzioni dal momento in cui restituiscono un risultato attraverso l'istruzione `return`

Il Perl non ne fa una distinzione linguistica (altri linguaggi invece sì) la diversità la fa il programmatore nel modo in cui le usa.

Per richiamarla basterà usare il simbolo `&` seguito dal nome della Sub, anche se ormai dalla versione 5.0.8 lo si può omettere.

```
&MenuPrincipale(); # Richiamano entrambi la stessa sub
MenuPrincipale();
```

E' ormai una regola non scritta quella di posizionare le sub in fondo al programma dopo il corpo principale del programma (main).

Il return può restituire non solo degli scalari, ma anche degli array e pure degli hash, dipende sia da come si dichiara la variabile che richiama la funzione che il ritorno della stessa, per cui:

```
@ordine_inverso = &Ordine_Inverso(1,2,3,4,5);

sub Ordine_Inverso {
    (@myarray) = @_;
    $lunghezza_array = @myarray-1;
    for($i=$lunghezza_array;$i>=0;$i--) {
        $elemento = $myarray[$i];
        push(@inverso,$elemento);
    }
    return @inverso;
}
```

## 7.2 Il Passaggio di Parametri

Quando passiamo dei parametri (valori) ad una sub, la loro gestione passa attraverso l'uso della Variabile Speciale @\_ (o Magic Array per i più fantasiosi) che li conterrà a mo' di lista.

```
$somma = &somma(1,2,3,4,5);

sub somma {
    $somma=0;
    foreach $num (@_) {
        $somma+=$num;
    }
    return $somma;
}
```

Questo modo di passare i parametri è necessario quando il loro numero può essere indefinito, ma quando si tratta di poche variabili le possiamo trasformare direttamente in scalari semplicemente con la sintassi:

```
($base,$altezza) = @_;
```

oppure utilizzando la notazione dell'array

```
$base = $_[0];
$altezza = $_[1];
```

**Se inseriamo più elementi di quelli che la sub si aspetta li perderemo senza rendercene conto.**

### 7.3 Il Passaggio di Array

---

Se anziché passare degli scalari volessimo usare degli array veri e propri? Ecco allora che riusciamo a tirare in ballo i Reference

```
$result = &miaFunzione(\@array);
```

mentre la Subroutine avrà la seguente sintassi:

```
sub miaFunzione {
    $array_ref = shift;
    @array = $$array_ref;
    foreach $elemento (@array)
    {
        print $elemento;
    }
}
```

Allo stessa maniera posso passare degli hash

```
$result = &miaFunzione(\@array, \%hash, $scalare);
```

```
sub miaFunzione {
    ($array_ref, $hash_ref, $scalare) = split;
    @array = $$array_ref;
    %hash = $$hash_ref;
}
```

La situazione non cambia nel caso di un return di più array o hash

```
return \%hash, \@array;
```

### 7.4 Librerie di File

---

Finora abbiamo visto la suddivisione logica, ovvero come creare subroutine e funzioni all'interno dello stesso file, ma per passare a quella fisica occorre introdurre il concetto di Libreria di File che nel Perl si avvale di due tipologie fondamentali:

- Il File Package
- Il File Modulo

## 7.5 Il File Package

---

Contiene variabili e funzioni riutilizzabili da qualsiasi altro programma che lo richiami. Fisicamente è un file sorgente .pl con l'intestazione `package` per ogni blocco che viene definito.

```
#!/usr/bin/perl
##### blocco_1.pl
package blocco_1;

sub funzione1{
    # Azione
}
sub funzione2{
    # Azione
}
```

Anche se è possibile mettere più blocchi di `package` all'interno di un unico file, è diventata prassi comune metterne solo uno.

Il programma che lo richiamerà userà la funzione `require` seguita dal nome del file. Se si omette il percorso lo cercherà nella directory corrente, altrimenti in una di quelle presenti in `@INC`

```
#!/usr/bin/perl
##### main_prg.pl

require "blocco_1.pl";

$alfa = funzione1();
$beta = funzione2();
```

## 7.5 Il File Modulo

---

È un file con estensione .pm e che obbligatoriamente contiene un solo `package` ed il nome del file deve essere lo stesso di quello indicato nel `package`.

Quando subroutine e funzioni diventano una parte separata dell'applicazione ed adattabile ad altri progetti, allora siamo in presenza di quello che può essere definito un modulo, ovvero un file separato che svolge delle funzionalità specifiche.

Si utilizza sempre l'istruzione `package` all'inizio dello script e `use` per richiamarlo dal programma che lo vuole utilizzare.

```
##### modulo.pm
package modulo;
```



```
sub funzione1{
    # Azione
}

sub funzione2{
    # Azione
}

##### programma.pl
#!/usr/bin/perl

use modulo;

$alfa = modulo::funzione1();
$beta = modulo::funzione2();
```

Se il file .pm non si trova nella directory corrente lo cercherà in uno dei percorsi presenti in @INC, se lo si vuole mettere in una sotto directory allora al momento dello USE dovrà essere indicata usando il simbolo :: come separatore (una sorta di backslash).

```
use Alexsoft::mia_libreria;
```

il modulo mia\_libreria.pm verrà cercato all'interno della dir Alexsoft

## 7.7 Require o Use ?

---

Dal punto di vista tecnico `require` lavora a run-time per cui se c'è un errore viene rilevato al momento in cui l'interprete lo incontra, mentre lo `USE` viene pre-compilato al momento dell'esecuzione dello script e quindi se c'è un errore si blocca prima, evitando la partenza stessa del programma che lo ha dichiarato.

Da un punto di vista della progettazione con l'uso del `require` dichiariamo un file sorgente esterno ma collegato all'applicazione principale come una procedura vera e propria, con lo `USE` invece vogliamo ampliare la nostra Libreria di Funzioni perché ad esempio il nostro programma deve implementare una certo algoritmo che altri hanno già progettato.

In conclusione il `require` ha più una valenza specifica del progetto mentre lo `USE` una più generica e riutilizzabile.

# 08

## Gestione dei File

### 8.1 I File di Testo

Quando parliamo di Files diamo per scontato che riguardano i File Ascii; il Perl, poi, nasce soprattutto per dare agli amministratori di sistema uno strumento in grado di manipolare i tanti files di Configurazione e di Log che albergano in un sistema Unix ed anche Windows (anche se ce ne sono molti di meno)

### 8.2 Apertura di un File

Qualunque sia la tipologia del file, la sintassi per aprirlo è

```
open (HANDLE , MODE , FILENAME) ;
```

L'**Handle** è il Canale attraverso cui viene gestito il file, viene anche definito *Descrittore del File*

Il **Mode** è la modalità di accesso al file

Il **FileName** è il nome del file da aprire completo di percorso (se omissso viene usato lo stesso su cui si trova lo script Perl)

#### Tabella Modo di Apertura di un File

Modo	Descr.	Read	Write	Append	New	Delete
<	Read	SI	NO	NO	NO	NO
>	Write	NO	SI	NO	SI	SI
>>	Append	NO	SI	SI	SI	NO
+<	Read/Write	SI	SI	NO	NO	NO
+>	Write/Read	SI	SI	NO	SI	SI
+>>	Append/Read	SI	SI	SI	SI	NO
COMANDO		NO	SI	/	/	/
COMANDO		SI	NO	/	/	/

La tabella riassume i simboli da utilizzare quando apriamo un file, ovvero se lo vogliamo aprire in **Read Only** - sola lettura e quindi impossibilitati ad eseguire delle modifiche - in **Write Only** - sola scrittura e quindi creando ex novo il file in sovrascrittura - in **Append** - aggiunge un record in coda - oppure in modalità combinate come da tabella.

La funzione `open()` ritorna 1 o 0 (Vero o Falso) a seconda che sia riuscita o meno ad aprire il file, può infatti accadere che per qualche motivo il file non possa essere aperto (file inesistente in caso di lettura, mancati attributi di accesso nel caso di sistemi Unix, ecc..)

```
if (open FILE,"<rubrica.dat") {
    # Leggo il file
}else {
    # Errore
    print "Impossibile Accedere al File Errore: $!\n";
}
```

Avrete notato due cose, la prima che il **Modo è stato inglobato nel Nome del File**, la seconda che ho utilizzato **la Variabile Speciale \$!** per la **descrizione dell'errore (\$ERRNO)**

Un altro modo più brusco e meno elegante di affrontare la cosa è usare l'istruzione `die()` per stroncare il programma stesso

```
open(FILE,"<rubrica.dat") ||
    die "Impossibile Accedere al File Errore: $!\n";
```

A mio parere il primo metodo è più leggibile e rientra nei canoni della programmazione strutturata, però è anche vero che molto spesso quando si tratta di script usa e getta la gestione degli errori è l'ultimo dei problemi.

### 8.3 Lettura di un File

Usando la semplice assegnazione ad uno scalare posso leggere il primo record (la riga che termina con un `\n` sui sistemi Unix e `\n\r` su quelli Windows) di un file di testo attraverso il canale di HANDLE.

```
$riga = <HANDLE>;
```

se invece utilizzo un array posso ricopiarvi l'intero contenuto del file

```
@testo = <HANDLE>;
```

ovviamente se si tratta di un file di configurazione con un numero limitato di righe, perché altrimenti conviene usare un ciclo avente la funzione eof() (End Of File) come condizione per leggersi tutto il file.

```
open(FILE,"<rubrica.dat") ||
    die "Impossibile Accedere al File Errore: $!\n";

$riga=0;
while (! eof(FILE)) {
    $buffer = <FILE>;
    $riga++;
    print "\n$riga = $buffer";
}
close FILE;
```

Esistono anche delle funzioni per leggere porzioni di files come read() ed altre che si spostano in determinate posizioni come seek()

```
read(HANDLE,$buffer,$TotaleByte);
```

\$buffer conterrà i primi \$TotaleByte del file HANDLE

```
seek(HANDLE,$ShiftByte,0);
```

ci sposteremo di tanti byte quanti indicati in \$ShiftByte dall'inizio del file (0 = indica l'inizio, 2 = la fine, 1 = nel mezzo)

## 8.4 Scrittura di un File

Se nella modalità in cui avete aperto il file c'è anche la scrittura, allora per registrare su file una qualsiasi stringa basta il semplice print seguito dal relativo HANDLE

```
print HANDLE "SCRIVO SUL MIO FILE\n";

# Apro il mio bel file di testo in Lettura/Scrittura
open(FILE,"+<rubrica.dat") || die "File non trovato!";
# Mi posiziono in fondo al file
seek(FILE,0,2);
# ed aggiungo l'ultimo record
print FILE "FINE DEL FILE";
close FILE;
```

in questo caso ho aperto un file in Read/Write simulando un Append.

Se lo avessi aperto direttamente in modalità >> mi sarei risparmiato il seek() ma questo dimostra come le funzioni in Perl sono talmente elastiche da poter essere usate in tanti modi diversi.

### 8.5 Operatori sui File

-r	Leggibile	If (-r "rubrica.dat")
-w	Scrivibile	If (-w "rubrica.dat")
-d	Directory	If (-d "/home/")
-f	File Regolare*	If (-f "rubrica.dat")
-T	File di Testo	If (-T "rubrica.dat")
-e	File esiste	If (-e "rubrica.dat")

\* Per Unix un File Regolare è un qualsiasi File che non è compreso nelle dir /dev e /sys

# Esempio di uso di operatori sui file per evitare  
# che l'apertura di un file non esistente generi un errore

```
$FileName = "rubrica.dat";
if (! -e $FileName) {
    print "Il File $FileName non esiste!!!\n";
}else {
    open(FILE,"<$FileName") || die "Errore: $!\n";
    @righe = <FILE>;
    close FILE;
}
```

# 09

# Espressioni Regolari

## 9.1 Abracadabra

Le RegExpr (o Regular Expression) rappresentano la parte magica del Perl (ma soprattutto di Unix) perché riescono in pochi caratteri a sintetizzare decine di righe di puro codice. Questa nomea deriva dal fatto che, agli occhi del profano, sono tanto potenti quanto nefaste. Sicuramente non contribuiscono alla leggibilità del codice e poi ci vuole un notevole sforzo per interpretare quelle scritte da altri. Ecco un chiaro esempio per farvi comprendere l'argomento di discussione:

```
/^[([\w\-\+\.\ ]+)([([\w\-\+\.\ ]+)([([\w\-\+\.\ ]+))$/
```

Di che si tratta? Beh questa RegExpr verifica la sintassi di un indirizzo e-mail, ma andiamo per gradi.....

## 9.1 Il Favoloso Mondo delle RegExpr

Le Regular Expression vivono in un mondo tutto loro, fatto di Operatori, Metacaratteri e Quantificatori. Gli Operatori rappresentano la struttura di questo micro-linguaggio.

### [Tab.1] Gli Operatori

Simbolo	Descrizione
\	Escape per caratteri speciali
.. ..	Alternanza (OR)
(..)	Gruppo
[..]	Classe di Caratteri (vedi Tab.2)
[^..]	Negazione della Classe di Caratteri
^	Cerca all'inizio della stringa
\$	Cerca alla fine della stringa
.	Trova qualsiasi carattere

Ogni espressione viene racchiusa dagli slash (/<regex>/)

I Metacaratteri rappresentano le scorciatoie alle Classi di Carattere.

**[Tab.2] I Metacaratteri**

Meta Carattere	Descrizione	Classe di Caratteri
\d	Numero	[0-9]
\D	Non Numero	[^0-9]
\s	Spazio	[\t\n\r\f]
\S	Non Spazio	[^\t\n\r\f]
\w	Alfanumerico	[a-zA-Z0-9_]
\W	Non Alfanum.	[^a-zA-Z0-9_]

Molto spesso non ci interessa sapere se una stringa è stata trovata n-volte in un testo, ma se quella istanza viene trovata solo n-volte ed allora che vengono in aiuto i Quantificatori.

I Quantificatori sono degli operatori che permettono di decidere quante volte un determinato carattere/stringa deve essere presente nella stringa di ricerca.

**[Tab.3] I Quantificatori**

Quantificatore	Descrizione
*	Trova 0 o più volte (max)
?	Trova 1 o 0 volte (max)
+	Trova 1 o più volte (max)
{n}	Trova esattamente n volte
{n,}	Trova almeno n volte (max)
{n,m}	Trova almeno n volte ma non più di m (max)
*?	Trova 0 o più volte (min)
+?	Trova 1 o più volte (min)
??	Trova 1 o 0 volte (min)
{n,}?	Trova almeno n volte (min)
{n,m}?	Trova almeno n volte ma non più di m volte (min)

### 9.3 Pattern Matching

E' il motivo per cui esistono le Espressioni Regolari, ovvero per trovare delle corrispondenze a delle sottostringhe (pattern) di ricerca.

```
if ($nome =~ m/Alessandro/) {
    <azione>
}
```

L'espressione logica `$nome =~ m/Alessandro/` assume il valore di vero nel caso in cui Alessandro sia contenuto in un punto qualsiasi di `$nome`

Il Matching (m) di una stringa è Case Sensitive per cui se non vogliamo avere problemi possiamo aggiungere una `i` alla chiusura dell'espressione:

```
if ($nome =~ m/alessandro/i) {
    print "Ciao Alessandro";
}
```

Proviamo ora ad aggiungere qualche Quantificatore per movimentare l'esempio:

```
/^(Alessandro){3}$/
```

Il matching ha successo solo se il pattern viene ripetuto 3 volte senza spaziatura e senza caratteri ne ad inizio ne a fine stringa. In pratica così:

```
AlessandroAlessandroAlessandro
```

```
# In questo caso Almeno 2 volte
/^(Alessandro){2,}$/
```

```
# Dalle 2 alle 4
/^(Alessandro){2,4}$/
```

```
# Almeno una volta
/^(Alessandro)+$/
```

Nella Tabella 3 oltre alla descrizione del Quantificatore indico tra parentesi se può innescare o meno (max o min) dei meccanismi di Greedy. Quando ad una RegExpr diciamo di cercare un pattern all'interno di una stringa viene applicato il metodo del "left-most longest match" (la stringa più lunga che soddisfa il pattern a partire dall'estrema sinistra) e quindi il massimo dell'ingordigia. Poco chiaro? Beh, facciamo un esempio:

```
/a{3,5}/
```

Non soddisfa solamente `aaaaa` ma anche `aaaaaaaaa` come pure



```
/a{3}/
```

che soddisfa sia aaa che aaaaaaaa

## 9.4 Pattern Substitution

Molto spesso al conseguimento di un matching può essere associata un'operazione di sostituzione o di traslitterazione.

Per Sostituzione s'intende cambiare un pattern con un altro, ad esempio:

```
$nome =~ s/Anna/Alessandro/g;
```

Ad ogni occorrenza (ricordate la g di global) del pattern "Anna" la sostituisco col pattern "Alessandro".

Per Traslitterazione invece s'intende convertire un set di caratteri con un altro, ed infatti il global non viene indicato.

```
$nome =~ tr/ab/cd/;
```

Significa che ogni "a" viene sostituita con una "c" mentre ogni "b" con una "d"

## 9.5 Esempi concreti

Mi rendo conto che le Espressioni Regolari possano sembrare un modo bizzarro di programmare ma molto spesso rappresentano una valida alternativa per risolvere problemi comuni, come ad esempio:

### 9.5.1 Data Entry

Controllare che l'input di un dato sia inserito nella giusta maniera può diventare seccante, ma con una semplice RegExpr

```
chomp ($input = <STDIN>);

if ($input =~ /^(..)\/(..)\/(....)/) {
    print "Data: $1/$2/$3";
}else {
    print "Data NON VALIDA - FORMATO GG/MM/AAAA";
}
```

Attenzione all'uso del carattere slash, occorre usarlo col backslash

Oppure se non vogliamo preoccuparci del CaseSensitive di un'opzione:

```
if ($input =~ /help/i) {
```

```
    print "Help On Line";
}
```

### 9.5.2 UpperCase/LowerCase

```
$stringa = "Le Cose cadono sempre ad Angolo Retto";
$stringa =~ tr/[a-z]/[A-Z]/; # Tutto Maiuscolo
$maiuscolo = $stringa;
$stringa =~ tr/[A-Z]/[a-z]/; # Tutto Minuscolo
$minuscolo = $stringa;
```

### 9.5.3 Trasformare un File DOS per UNIX

Rimuovendo tutti i Carriage Return \r

```
$buffer = <FILEIN>; # Questo l'abbiamo visto nel Capitolo 8
$buffer =~ s/\r//g;
```

## 9.6 L'Enigma Svelato

Finisco come avevo iniziato, ovvero con la stringona che controllava un indirizzo email; proviamo a guardarla alla luce di quanto detto fino ad ora...

```
/^[([\w\-\+\.\.]+)@([\w\-\+\.\.]+)\.([\w\-\+\.\.]+)$/
```

Non vi è un po' più familiare così?

```
/^[([\w\-\+\.\.]+)@([\w\-\+\.\.]+)\.([\w\-\+\.\.]+)$/
```

Notiamo immediatamente che i tre gruppi in realtà eseguono il medesimo controllo, ovvero quello della presenza di almeno (+) un qualsiasi carattere alfanumerico (il metacarattere \w) o dei simboli più, meno e punto.

Un indirizzo internet deve avere all'inizio (ricordate il ^) una qualsiasi stringa seguita dalla chiocciola (at per gli inglesi) dopodichè il suo dominio e come ultima cosa (ritorniamo all'ancora \$) dopo il punto il dominio di primo livello. Per cui a questa condizione vanno bene tutti questi indirizzi:

```
alex@ciccio.net
hallex.9000@mira.uk.org
zuffolo-mania@pluto.biz.com
```

e verranno scartati altri come:

```
alex@ciccio
mira.uk.org
@pluto.biz.com
```

# 10

## Le Funzioni Standard

### 10.1 Una Breve Presentazione

---

Quelle che mi appresto ad elencarvi rappresentano le funzioni standard del Perl, ovvero quelle funzioni già incluse nell'interprete e quindi parte del linguaggio. Alcune di queste funzioni le abbiamo già viste in precedenza, per le altre c'è l'assoluta novità. Lo scopo di questo capitolo è raggrupparle per funzione in modo da associarle alle azioni per cui sono state progettate. Queste non rappresentano tutte le funzioni ma quelle più comuni ed utilizzate.

### 10.2 Funzioni Aritmetiche

---

Funzione	Descrizione
abs(espr)	Valore assoluto dell'espressione
cos(espr)	Coseno trigonometrico dell'espressione
exp(espr)	Esponenziale (un numero elevato a espr)
int(espr)	Valore Intero
log(espr)	Logaritmo naturale (in base e) di espr
rand(espr)	Valore casuale (non intero) tra 0 ed espr
sin(espr)	Seno Trigonometrico di espr
sqrt(espr)	Radice Quadrata di espr

Oltre alle funzioni aritmetiche vere e proprie (valore assoluto, seno, coseno, radice quadrata, ecc..) che hanno un utilizzo prettamente scientifico ne esistono altre che con un po' di fantasia possono essere adattate a qualunque problematica. Ad esempio:

```
$totale_amici = @amici;
$numero_casuale = int(rand $totale_amici);
print "Oggi esci col tuo amico $amici[$numero_casuale]";
```

la funzione rand non ritorna un intero ma un long (virgola mobile) per cui dobbiamo mantenere la sola parte decimale. Se però il range dei numeri non

parte da 0 ma da 1 perché non si tratta di un array ma di un database, allora dovremmo aggiungere 1.

```
$numero_casuale = int(rand $totale_amici)+1;
```

### 10.3 Funzioni di Conversione

Funzione	Descrizione
chr(n)	Carattere Ascii corrispondente al valore decimale n
hex(n)	Valore Decimale del numero esadecimale n
oct(n)	Valore Decimale del numero ottale n
ord(\$car)	Codice Ascii del carattere \$car
sprintf(\$format,\$var)	Formatta la stringa \$var

```
print chr(65);
print ord("L");
print hex(255);
```

Lo `sprintf` è simile al `printf` (stessi Specificatori di Campo, vedi Capitolo 6) con la sola differenza che anziché spedire un output in un Handle lo invia ad una variabile.

```
$hex = sprintf "%X", $decimale;
```

Restituisce un intero esadecimale con lettere maiuscole (%x per averle minuscole)

In Perl non esiste una funzione per arrotondare un valore frazionato, l'utilizzo dell' `int` provoca il semplice troncamento della parte decimale, mentre lo `sprintf` può fare al caso nostro

```
$importo = sprintf "%.0f", $importo;
```

L'arrotondamento avverrà o per eccesso o per difetto a seconda della cifra e al numero di decimali indicati

### 10.4 Funzioni su Stringhe

Funzione	Descrizione
chop(\$var)	Elimina l'ultimo carattere dal \$var
chomp(\$var)	Elimina l'ultimo carattere solo se è \n
eval(espr)	Valuta l'espressione Perl espr
index(\$stringa,\$pattern)	Indica la posizione di \$pattern all'interno

	di \$stringa
length(\$var)	Lunghezza della stringa \$var
lc(\$var)	Restituisce \$var in caratteri minuscolo
rindex(\$stringa,\$pattern)	Come Index ma partendo da destra anziché da sinistra
substr(\$var,\$offset,\$len)	Estrare da \$var a partire dalla posizione \$offset \$len caratteri
uc(\$var)	Restituisce \$var in caratteri maiuscoli
join(\$sep,@array)	Concatena gli elementi dell'@array in un'unica stringa usando il contenuto di \$sep come carattere di separazione

Ecco alcuni esempi di utilizzo delle funzioni per le stringhe:

```
$quotes = "Quando una cosa può andare male lo farà sicuramente";
print "\n" . substr($quotes,7,3) ;
print "\n" . substr($quotes,-7,3) ;
print "\n" . index($quotes, "una");
print "\n" . uc($quotes)
print "\n" . index($quotes, "casa");
```

```
una
ram
7
QUANDO UNA COSA Può ANDARE MALE LO FARÀ SICURAMENTE
-1
```

A questo punto possiamo notare che:

- Se l'offset è negativo il conteggio partirà da destra anziché da sinistra;
- Il conteggio dei caratteri all'interno di una stringa parte dallo zero se avviene da sinistra mentre 1 se da destra;
- I caratteri speciali (le lettere accentate o i simboli) non possono subire un processo di UpCase o LowCase;
- La posizione -1 indica che il pattern indicato non è stata trovato all'interno della stringa;

Finiamo col join utile per trasformare un array in uno scalare (e quindi con il limite dei 256 caratteri).

```
@nomi = ("lun","mar","mer","gio","ven","sab","dom");
$elenco = join("|",@nomi);
print "\n". $elenco;
```

```
lun|mar|mer|gio|ven|sab|dom
```

## 10.5 Funzioni su Array ed Hash

Funzione	Descrizione
<code>delete \$array[\$index]</code>	Cancella dall'array l'elemento \$index (funziona anche per l'hash)
<code>each(%hash)</code>	Estrae la coppia Chiave e Valore dall'hash
<code>exists \$array{chiave}</code>	Verifica l'esistenza di una chiave nell'hash
<code>grep /regexpr/,@array</code>	Restituisce gli elementi dell'array che soddisfano una certa regexpr (Espressione Regolare)
<code>keys %hash</code>	Restituisce una lista con le chiavi dell'%hash
<code>map /regexpr/@array</code>	Esegue una regexpr o una funzione per ogni elemento dell'@array
<code>pop(@array)</code>	Restituisce ed elimina l'ultimo elemento dell'@array
<code>push(@array,\$elem)</code>	Inserisce \$elem in coda all'array
<code>reverse(@array)</code>	Restituisce l'@array in ordine inverso
<code>shift(@array)</code>	Restituisce ed elimina il primo elemento dall'array
<code>sort(@array)</code>	Ordina gli elementi dell'array
<code>splice(@array,offset,tot,lista)</code>	Rimuove gli elementi dell'array a partire dall'indice offset per tot elementi e li rimpiazza con gli elementi della lista
<code>split(\$pattern,\$stringa)</code>	Genera un array da una \$stringa in base al carattere di separazione \$pattern
<code>unshift(@array,elem)</code>	Aggiunge elem (scalare o array) in testa all'@array
<code>values %hash</code>	Restituisce un array con i valori degli elementi dell' %hash

Oltre a quelle per la gestione dell'array viste nel Capitolo 4 esistono altre importanti funzioni per la manipolazione degli array; iniziamo con `grep` che prende il nome da una famosa utility Unix per trovare degli elementi all'interno dei file (un po' come il Find dell'MsDos), qui invece si tratta di cercare una determinata stringa all'interno di un array. Usando la notazione delle Espressioni Regolari:

```
@nomi = ('Alessandro','Alberto','Alessio','Alessandra');
@trovati = grep /Ales/,@nomi;
print "\n @trovati";
```

Alessandro Alessio Alessandra

`map` invece permette di eseguire una certa espressione (o funzione) per ogni elemento dell'array; in questo caso non bisogna confondere il simbolo `@`

(usato in combinazione col backslash per indicare il simbolo e non una struttura dati) da quello che identifica l'array vero e proprio.

```
@nuovi = map {"$_\@tiscali.it"}@trovati;
print "\n @nuovi";
```

```
Alessandro@tiscali.it Alessio@tiscali.it Alessandra@tiscali.it
```

ho aggiunto un dominio di posta elettronica per ogni elemento `$_` dell'array `@trovati`. Potrebbe risultare molto utile se, all'interno dell'array, ci fossero dei valori numerici su cui dover eseguire dei veri e propri calcoli. Nell'esempio proposto incrementiamo di uno ogni elemento dell'array `@numeri`;

```
@numeri = (10,20,30,40,50);
@somma1 = map{$_+1}@numeri;
```

`reverse` permette di creare un nuovo array di ordine inverso rispetto a quello in input;

```
@numeri = (1,2,3,4,5);
@contrario = reverse(@numeri);
print "\n @contrario";
```

```
5 4 3 2 1
```

Molto interessante l'abbinamento con il `sort`, infatti se quest'ultimo permette di ordinare un array in modo discendente (dal più grande al più piccolo), col `reverse` è possibile farlo nel modo ascendente.

```
@numeri = (340,122,90,43,78);
@ord_asc = sort(@numeri);
@ord_dis = reverse(@ord_asc);
```

Il `sort` permette l'ordinamento guardando i primi caratteri dell'elemento di un array, se però abbiamo un hash occorre prestare attenzione perché a quel punto verrà ordinata la chiave (key) e non il valore (value). La stessa cosa è valida per tutte le altre funzioni che richiedono un array come parametro di input. Un esempio classico è un hash contenente la classifica di serie A; come chiave abbiamo il nome della squadra e come valore il suo punteggio:

```
%serieA = (
    'Milan' => 79,
    'Juventus' => 86,
    'Inter' => 72,
    'Roma' => 45,
    'Lazio' => 44,
    'Udinese' => 62,
    'Sampdoria' => 61);
```

```
%classifica = sort(%serieA);
print "\n @classifica";
foreach $value (sort {$serieA{$a} cmp $serieA{$b} } keys %serieA)
{
    print "\t\t$value $serieA{$value}\n";
}
```

Quella che viene considerata la funzione più potente per la manipolazione degli Array è in realtà quella che viene meno utilizzata, infatti lo `splice` fa quello che normalmente farebbe l'uso congiunto di altre quattro funzioni precedentemente documentate: `push`, `pop`, `unshift` e `shift`. Ecco un paio di esempi:

<code>push(@arr,@newarr)</code>	<code>splice(@arr,\$#newarr+1,0,@newarr)</code>
<code>\$elem = pop(@arr)</code>	<code>\$elem = splice(@arr,-1)</code>
<code>unshift(@arr,@newarr)</code>	<code>splice(@arr,0,0,@newarr)</code>
<code>\$elem = shift(@arr)</code>	<code>\$elem = splice(@arr,0,1)</code>
<code>\$arr[\$i] = \$elem</code>	<code>splice(@arr,\$i,1,\$elem);</code>

Ma la sua potenza sta nel fatto di potere lavorare non solo su di un singolo elemento ma su di un intero intervallo.

```
# Rimuovi due elementi a partire dall'indice 2 (compreso)
splice(@array,2,2);
# Sostituisci due elem. a partire dall'indice 2 con un nuovo array
splice(@array,2,2,@newarray);
```

Infine lo `split` ci permette di spalmare su di un array una stringa che separi gli elementi con un simbolo comune.

```
@nomi = split(",","Alessandro,Giacomo,Alfredo,Giovanni");
```

Avrò un array riempito con i quattro elementi separati dalla virgola (che in questo caso rappresenta il pattern)



## 10.6 Funzioni su File e Directory

Funzione	Descrizione
<code>chmod(\$modo,@array)</code>	Cambia i permessi su file specificati nella lista (unix)
<code>chown(\$user,\$group,@array)</code>	Cambia il proprietario ed il gruppo dei file specificati nella lista (unix)
<code>glob(\$dir)</code>	Restituisce un array contenente il nome dei file che hanno soddisfatto la condizione \$dir (usa le wildcard *)
<code>mkdir(\$dir,\$modo)</code>	Crea la directory con i permessi indicati in \$modo
<code>truncate(\$filename,\$dim)</code>	Tronca il \$filename alla dimensione \$dim
<code>rename(\$vecchio,\$nuovo)</code>	Rinomina il vecchio file col nuovo
<code>rmdir(\$dir)</code>	Elimina la directory (vuota) \$dir
<code>unlink(@array)</code>	Elimina i file specificati nella lista
<code>system("comando","par")</code>	Esegue un comando esterno

Alcune funzioni su file e directory hanno come interlocutore un sistema basato su Unix (Linux/Aix/FreeBSD/Solaris ec...) per cui non hanno molto senso in sistemi Microsoft come Windows.

Un esempio è `chmod` che serve a cambiare gli attributi di un file alla maniera di Unix, ovvero la triade composta da Owner (lo user proprietario del file) il Group (il gruppo a cui lo user appartiene, solitamente users) e Other (tutti gli altri che non sono né il proprietario né appartenenti al gruppo). Della stessa pasta abbiamo `chown` che permette all'utente amministratore (root) di cambiare di proprietario e di gruppo a chicchessia.

```
@dir = glob("*.mp3");
chown('alexsoft','users',@dir);
chmod('750',@dir);
```

Per gli altri comandi cross-platform invece abbiamo `mkdir`, un classico per creare una nuova directory, `rmdir` per cancellarla (ma non ci devono essere files al suo interno) `rename` per rinominare un file e `unlink` (una novità per quello che riguarda il nome ma non il suo funzionamento) per cancellarlo.

Tutte queste variabili ritornano true o false a seconda dell'esito. Ma attenzione, se io voglio creare una cartella già esistente riceverò sempre un false come esito anche nel caso in cui non ho i diritti per creare una directory. Quindi occhio ad usare blocchi condizionali (IF-ELSE) che potrebbero saltare parti di codice. Ad esempio:

```
if (!mkdir("work") {
    print "Errore Creazione Directory";
}else {
    print "OK Directory Creata";
    &Fai_Tutto();
}
```

Finiamo con `truncate` per "troncare" (appunto) un file ad una dimensione impostata e `glob` che abbiamo visto nell'esempio precedente per riempire un array di nomi di files che soddisfano la condizione dettata in input.

```
truncate("miofile.txt",1024);
```

Mantiene solamente i primi 1024 byte di miofile.txt

### 10.7 Funzioni di Input/Output

Funzione	Descrizione
<code>close HANDLE</code>	Chiude un file precedentemente aperto
<code>eof(HANDLE)</code>	Restituisce Vero quando incontra la Fine del File (End of File)
<code>getc(HANDLE)</code>	Legge un carattere dall'HANDLE indicato
<code>open(HANDLE, \$modo,\$filename)</code>	Apre il file e gli associa il nome logico HANDLE
<code>print HANDLE \$stringa</code>	Scrive \$stringa sul file HANDLE
<code>read(HANDLE, \$var,n,offset)</code>	Legge n byte dal file a partire dalla posizione offset e li memorizza in \$var
<code>seek(HANDLE,\$pos)</code>	Posiziona il puntatore all'interno del file
<code>tell(HANDLE)</code>	Restituisce la posizione del puntatore all'interno del file

Queste funzioni sono state ampiamente documentate nel Capitolo riguardante la Gestione dei Files. Le uniche funzioni rimaste fuori sono `getc` e `tell`

`getc()` legge un carattere per volta dall'HANDLE indicato

```
open (FILEHANDLE, "< test.txt") or die "Errore: $!";
while(<FILEHANDLE>) {
    $char = getc(FILEHANDLE);
    print $char;
}
close(FILEHANDLE);
```

tell() dice (appunto) in che posizione si trova il puntatore del file, molto spesso viene usata in abbinamento a seek che sposta fisicamente il puntatore (numero di byte letti) per capire in che posizione si trova all'interno del file.

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

### 10.8 Funzioni di Data e Ora

Funzione	Descrizione
time	Resituisce la data attuale in formato TIMESTAMP
localtime(\$timestamp)	Resituisce l'array Time/Date come da Tabella
gmtime(\$timestamp)	Restituisce l'array come localtime ma in Universal Time (ex Greenwich Mean Time)

time ritorna l'orario in formato TimeStamp, ovvero il numero di secondi trascorsi all'inizio dell' 01-01-1970 (la data simbolo dei sistemi Unix)

#### Tabella Array Time/Date di Perl

Indice	Descrizione
0	Secondi (0-60)
1	Minuti (0-59)
2	Ore (0-23)
3	Giorno del Mese (1-31)
4	Mese dell'anno (0-11)
5	Anno (+1900)
6	Giorno della Settimana (0-6)
7	Giorno dell'anno (1-366)
8	Ora Legale (true se è attiva)

localtime e gmtime danno il medesimo risultato, ovvero riempiono un array (o una sequenza di scalari) convertendo un TimeStamp in Time/Date.

La differenza è che localtime restituisce queste informazioni in base all'orario locale della macchina su cui viene lanciato (e qui dipende dal Sistema Operativo installato), mentre gmtime lo fa in termini di GMT (Greenwich Mean Time, è Londra a dettare le regole) meglio conosciuto come UTC (Coordinated Universal Time) o Zulu Time per i militari.

```
( $sec, $min, $hh, $dd, $mm, $yy, $week, $totd, $legal ) = localtime(time);
$yy+=1900;      # 1900 è l'anno zero
$mm+1;         # gennaio è il mese zero
printf("\nOggi è %2d/%2d/%4d", $dd, $mm, $yy);
@days_week=( 'dom', 'lun', 'mar', 'mer', 'gio', 'ven', 'sab' );
printf("\nGiorno della settimana: %s", @days_week[$week]);
printf("\nSono passati %d giorni dall'inizio del %d", $totd, $yy);
```

Per quanto possa essere bizzarro il TimeStamp è uno strumento utile se vogliamo sommare un certo numero di giorni ad una data. Nell'esempio moltiplico il numero di giorni che mi interessa sommare per il numero di secondi al giorno.

```
$secondi_giorno = 86400;
$oggi = time();
$oggi = time();
@oggi = localtime($oggi);
@week = localtime($oggi+(7*$secondi_giorno));
printf("Oggi è %2d/%2d/%4d",@oggi[3],@oggi[4],@oggi[5]);
printf("Tra una settimana %2d/%2d/%4d",@week[3],@week[4],@week[5]);
```

# 11

## Il Debugging

### 11.1 Errare Humanum Est

---

Scrivere codice perfetto senza il minimo errore è un qualcosa di impossibile, quando si parla di debug del codice non ci si riferisce ad errori formali o di sintassi perchè a quello ci pensa l'interprete Perl, prima di mandare in esecuzione uno script ne verifica le strutture sintattiche ed abortisce nel caso di errori dando il numero della riga e perfino una sua spiegazione sui probabili motivi che hanno portato all'errore.

### 11.2 Prevenire è Meglio che Curare

---

Esistono alcuni accorgimenti che, un buon programmatore Perl, deve prendere in considerazione quando si appresta a scrivere del codice, per evitare in futuro, di perdere troppo tempo prima di capire che un semplice problema di variabili mal dichiarate blocchi l'intero progetto.

Una delle prime regole è l'utilizzo dello switch `-w` (warnings) quando lanciamo il Perl

```
perl -w nomesorgente.pl
```

Da Unix basta includerlo nello she-bang iniziale

```
#!/usr/bin/perl -w
```

oppure

```
use warnings;
```

e l'immaneabile

```
use strict;
```

Cosa chiediamo al Perl?

- Di segnalare a video ogni warning, ovvero un errore non bloccante ma che può indicare del codice a rischio di bug;
- Di generare un errore se si utilizza un identificatore che non sia una sub precedentemente dichiarata
- Di genera un errore di compilazione se trova una variabile non dichiarata con `my`

Il primo è l'uso dello switch `-w` o `use warnings` mentre per tutti gli altri la dichiarazione iniziale della direttiva al compilatore `use strict`

Mandando in esecuzione questo script

```
use strict;
$alfa = "beta";
$beta = Opera();
print "\n $alfa $beta";
```

Avremo un blocco per le seguenti motivazioni:

```
Global symbol "$alfa" requires explicit package name at debug.pl line 4.
Global symbol "$beta" requires explicit package name at debug.pl line 5.
Global symbol "$alfa" requires explicit package name at debug.pl line 7.
Global symbol "$beta" requires explicit package name at debug.pl line 7.
Execution of debug.pl aborted due to compilation errors.
```

Obbligare a dichiarare le variabili col `my` ci segnala tempestivamente se abbiamo sbagliato a digitare il nome di una variabile in qualche parte del codice. Ad esempio:

```
my $Nome="ciccio";
print $Name;
```

mi dirà che c'è una variabile globale `$Name` che non è stata dichiarata e al quel punto dovrei capire qualcosa....

### 11.3 Il Debugger

Lo switch `-d` permette di debuggare il codice in linea di comando, ma esistono dei tools che permettono di fare la stessa cosa in modo un po' più visuale. Fondamentalmente un debugger permette due tipi di operazioni:

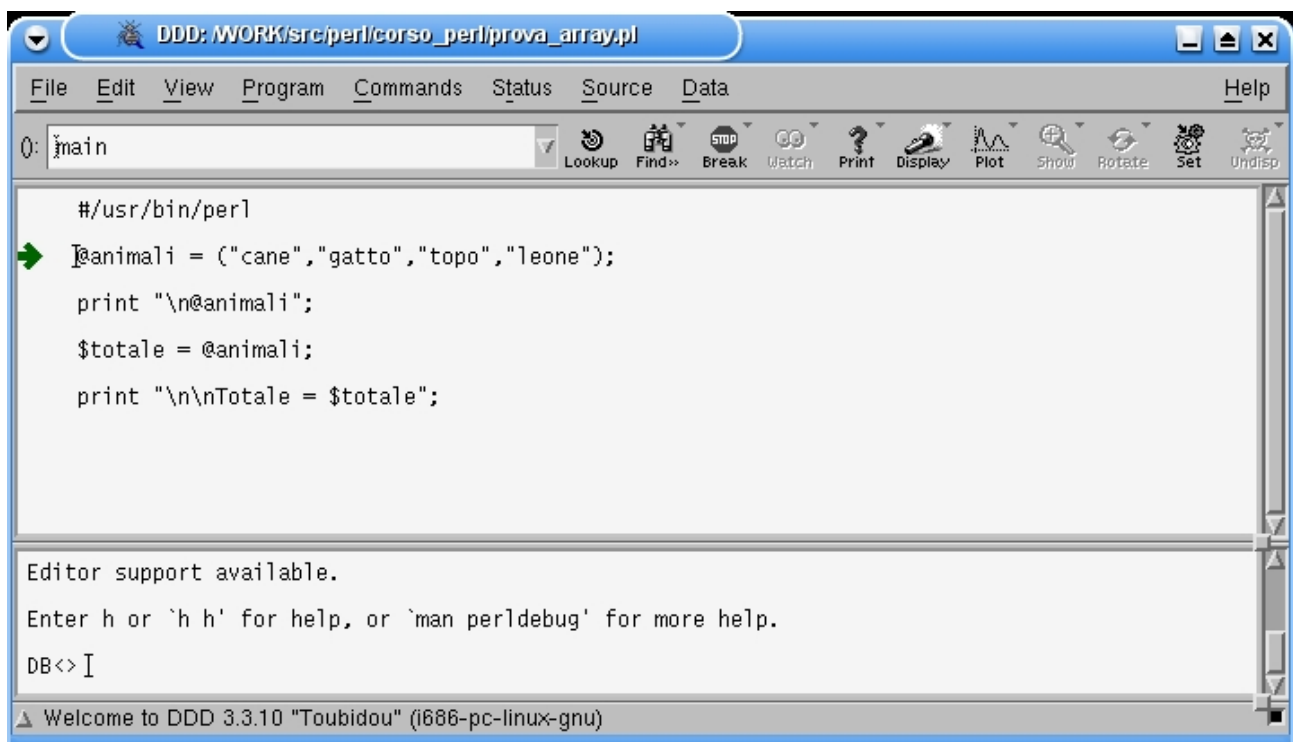
1. Impostare dei **BreakPoint** (punti in cui interrompere uno script) in punti critici (se si ha già un'idea di massima) per scoprire dove si annida la falla
2. Partire dall'inizio in **Step Over** per eseguire lo script una riga per volta fino alla fine

Dal momento in cui stoppiamo il Debug abbiamo la possibilità di analizzare il conseguente stato dello script e relativo:

- Output della console
- Contenuto delle variabili
- Lista dei moduli caricati

Per Windows di OpenSource c'è il buon PerlIDE che oltre ad essere un buon ambiente per scrivere codice può servire per le classiche operazioni di debug appena descritte.

In Linux c'è anche qualcosa di meglio, il Data Display Debugger o DDD per gli amici ([www.gnu.org/software/ddd/](http://www.gnu.org/software/ddd/)) che attraverso un'interfaccia grafica permette di eseguire tutte quelle operazioni per testare parti di codice che normalmente diventerebbero troppo macchinose fatte a linea di comando.



## 11.4 In Conclusione

Scrivere del buon codice in tempi ragionevoli dipende oltre dagli strumenti a disposizione del programmatore, anche dal metodo di lavoro che si persegue. Per questo sono necessarie una serie di regole da seguire per evitare di incorrere nei problemi più comuni:

- Scegliete Identificatori (variabili/sub/moduli) mnemonici; chiamare una scalare che contiene un contatore \$pippo non è molto chiaro
- Scrivere le Costanti in MAIUSCOLO
- Scrivere le variabili dichiarate con my o local in minuscolo

- Le sub devono iniziare con la MAIUSCOLA mentre le funzioni devono essere tutte in minuscolo
- Le Espressioni Regolari spaziate
- Pensate alla riutilizzabilità del codice; se iniziate a scrivere un programma con l'intento di riutilizzarlo per altre problematiche eviterete i classici errori del codice "usa e getta"

### Domande di Fine Modulo

---

1. Differenza tra Scalari, Array ed Hash
2. Cosa s'intende per "Interpolazione di Stringa" ?
3. Qual è l'elemento di identificazione per la gestione dei file ?



# 12

## I Moduli Standard

### 12.1 Introduzione

---

I Moduli Standard (o Core Modules) sono dei file esterni con estensione .pm che vengono installati insieme al Perl fin dalla versione 5.005. In realtà una volta che abbiamo installato un modulo esterno diventa di fatto un Modulo Standard, si tratta per lo più di una distinzione iniziale per capire se una certa funzione ha bisogno di un modulo esterno oppure ce l' ha già in casa.

### 12.2 I Moduli Pragma

---

Vengono raggruppati insieme ai Moduli Standard ma sono dei moduli che hanno effetto sulla fase di compilazione (direttive, ovvero suggeriscono al compilatore la strada da percorrere), per convenzione **questi moduli hanno il nome tutto minuscolo** e sono i seguenti:

constant	Dichiara costanti
Diagnostics	Come warnings ma con più informazioni
Integer	Usa Integer piuttosto che floating quando assegno un numero
Open	Settare Input/Output di default
Strict	Pone delle limitazioni ai costrutti (variabili e sub) non sicuri
Warnings	Stampa ogni warning del Perl (equivalente a perl -w)

### 12.3 I Moduli Standard

---

Nella documentazione ufficiale, all'indirizzo [perldoc.perl.org/index-modules-A.html](http://perldoc.perl.org/index-modules-A.html) troverete un'ampio elenco di Moduli Standard (mischiate ai Moduli Pragma) che per semplicità li raggrupperò per categoria spiegando solamente quelli più utili ed importanti.

<b>AnyDBM_File</b>	Gestione File DBM
AutoLoader	Carica le funzioni solo su richiesta
Autosplit	Suddivide un modulo per il caricamento automatico
B	Strumenti e compilatore Perl
Benchmark	Benchmark del codice
Carp	Genera messaggi di errore
<b>CGI</b>	<b>Common Gateway Interface (Cap.18)</b>
Config	Accede alle informazioni di configurazione del Perl
CPAN	Gestione per il download ed installazione dei moduli CPAN
Cwd	Directory di lavoro corrente
Data::Dumper	Struttura dati Perl sotto forma di stringhe
DB_File	Accede al DB di Berkeley
DynaLoader	Caricamento dinamico automatico di moduli Perl
Env	Importa variabili d'ambiente
Errno	Costanti errno.h di sistema (UNIX)
Exporter	Importazione di Moduli
ExtUtils	Comandi Unix Comuni
<b>File</b>	<b>Gestione dei File</b>
<b>Getopt</b>	<b>Gestione degli switch a riga di comando (Cap.20)</b>
IO	Gestione Input/Output
IPC	IPC System V
Math	Funzioni Matematiche
<b>Net</b>	<b>Interfaccia di Rete (Cap.19)</b>
Pod	Moduli POD (Plain Old Documentation)
SelfLoader	Carica funzioni solo su richiesta
<b>Shell</b>	<b><u>Esegue comandi di shell Unix</u></b>
Socket	Gestione dei Socket di Rete
<b>Sys::Hostname</b>	<b>Restituisce l'Hostname (Cap.19)</b>
Term	Interfaccia Terminale
<b>Text</b>	<b>Gestione del Testo</b>
Thread	Supporto Multithreading
Tie	Definizioni di classi base
Time	Gestione Ora simile al localtime()

I Moduli Standard, per convenzione, hanno tutti l'iniziale maiuscola, per differenziarsi dai Moduli Pragma.

12.4 use File::\*

File::BaseName	Parsing del nome di un file
File::Compare	Differenze tra due file
File::Copy	Copia File
File::Find	Trova File
File::stat	Statistiche di un file

Con File::BaseName abbiamo una serie di funzione per il parsing del percorso di un file :

- basename
- dirname
- fileparse
- fileparse\_set\_fstype(OS)
  - VMS
  - MSWin32
  - MSDOS
  - AmigaDOS
  - Os2
  - MacOS

```
use File::BaseName;
@array_ext=('doc','exe','pl','bat');
$filename = "/home/alexsoft/documenti/corsoperl.doc";
($nomefile,$path,$ext)= fileparse($filename,@array_ext);
print "\n" . basename($filename);
print "\n" . dirname($filename);
print "\n" . $ext;
```

```
/home/alexsoft/documenti/
corsoperl.doc
doc
```

fileparse() estrai tre nomi: il primo è il file, il secondo è il path ed il terzo è il suffisso se questo è presente in @array\_ext

Il fileparse\_set\_fstype(\$OS) va posto all'inizio ed indica al parser qual è il Sistema Operativo di riferimento (se lasciato vuoto è Unix)

File::Find() merita un piccolo approfondimento; la sua funzione è quella di cercare i file che corrispondono ad un'espressione particolare indicata in una subroutine. Ha un funzionamento molto simile all'omonimo comando Unix.

```
use File::Find;
$dirStart = "D:/mp3";
find \&deep,$dirStart;
```

```
sub deep{
  if (-f) {
    $fileName = $File::Find::name;
    print "\n$fileName";
  }
}
```

Nel nostro esempio `find` interroga la subroutine `deep` (creata ad-hoc) che gli dice di stampare i nomi di tutti i file che incontra (`-f`) a partire dalla `$dirStart` (che può anche essere un array di percorsi).

Oltre alla funzione `$File::Find::name` esiste anche la `$File::Find::dir` (la directory corrente)

Le funzioni `File::Copy` e `File::Compare` possiamo vederle nell'esempio:

```
use File::Copy;
use File::Compare;

$file1="d:\\mp3\\greenday-she.mp3";
$file2="h:\\mp3\\greenday-she.mp3";
if (compare($file1,$file2) {
  print "\nI File Sono Uguali";
}else{
  copy($file1,$file2);
}
```

`cmp()` viene usato spesso al posto di `compare()`

Con `File::stat` abbiamo una serie d'informazioni riguardo al file indicato:

Campo	Significato	Campo	Significato
<b>dev</b>	Numero di Dispositivo del FileSystem	<b>ino</b>	Numero di I-node
<b>mode</b>	Diritti di accesso	<b>nlink</b>	Numero di collegamenti al file
<b>uid</b>	ID dell'owner	<b>gid</b>	ID del gruppo
<b>rdev</b>	Identificativo di dispositivo	<b>size</b>	Dimensione del file in byte
<b>atime</b>	Data Ultimo Accesso	<b>mtime</b>	Data Ultima modifica
<b>ctime</b>	Data variazione Inode	<b>blksize</b>	Dimensioni di blocco predefinite per I/O file system
<b>blocks</b>	Numero di blocchi allocati		

Diciamo che non tutti i campi sono compatibili con Win32, solo quelli in grassetto, inoltre le date vengono sempre espresse in TimeStamp.

```
$stats = stat($filename);
$size = $stats->size;
```

### 12.5 use Text::\*

Text::Soundex	Cerca pattern che sono simili
Text::Wrap	Racchiude il testo in un paragrafo
Text::Tabs	Espande o meno le tabulazioni

Text::Soundex implementa l'algoritmo Soundex che rielabora una parola (utile per i cognomi) in una forma compressa che si avvicina al suono della parola in lingua inglese. Una sorta di Hashing del nome.

```
use Text::Soundex;
$code = soundex($nome);
```

Text::Wrap formatta un paragrafo secondo il numero di colonne ed il rientro dal margine sinistro della prima riga e delle successive.

```
use Text::Wrap;
$rigatesto="E'Sempre Colpa del Compagno - Prima Legge del Bridge";
$Text::Wrap::columns=35;
print wrap(" "x5," "x2,$rigatesto);
```

E'Sempre Colpa del Compagno -  
Prima Legge del Bridge

columns indica a che colonna deve andare a capo mentre wrap() formatta la stringa in \$rigatesto rientrando di cinque spazi nella prima riga e di 2 in tutte le altre.

Text::Tabs fornisce due funzioni; expand() sostituisce ciascun carattere di tabulazione con un numero equivalente di spazi, unexpand() invece sostituisce una sequenza di \$tabstop con un carattere di tabulazione

### 12.6 use Shell

Consente di invocare le utility di shell Unix come se fossero delle sub del Perl. I comandi che si voglio utilizzare vanno posti dopo la funzione qw tra le parentesi tonde. **Non funziona sotto Win32**

```
use Shell qw(date cp ps);
```

```
# date è il comando Unix  
$date = date();
```

Gli argomenti (compresi gli switch) vengono passati sotto forma di stringhe.

```
cp("-p", "/etc/groups", "/tmp/groups");
```

# 13

# I Moduli CPAN

## 13.1 Il Repository CPAN

Il Comprehensive Perl Archive Network è un archivio in rete che rappresenta ciò che la comunità Perl ha messo a disposizione agli sviluppatori in termini di utility, moduli e documentazione. La sua potenza consta nell'avere tantissimi mirror (repliche del server) dislocati in tutto il mondo. Se vogliamo avere un'idea dell'enormità del repository (deposito di file) andiamo su sito ufficiale [www.cpan.org](http://www.cpan.org) e proviamo ad elencare tutti i moduli che ha a disposizione per rimanerne sbalorditi.



Ripeto che si tratta di un qualcosa di completamente gratuito ed opensource che permette al programmatore di essere immediatamente produttivo su qualsiasi problematica da affrontare poiché c'è già stato qualcuno che l'ha risolta e ha reso disponibile a tutti il proprio lavoro.

### 13.2 Com è Organizzato il CPAN?

Il materiale all'interno del CPAN è diviso per categorie (moduli Perl, documentazione, scripting, autori, ec...) ed ogni categoria può essere o meno collegata ad un' altra. Ad esempio un link ad un modulo grafico scritto da un autore appare sia nel CPAN dei moduli che nell'area degli autori. La maggiorparte del materiale è distribuito sottoforma di file compressi .tar o .gzip (il popolare sistema di archiviazione di Unix).

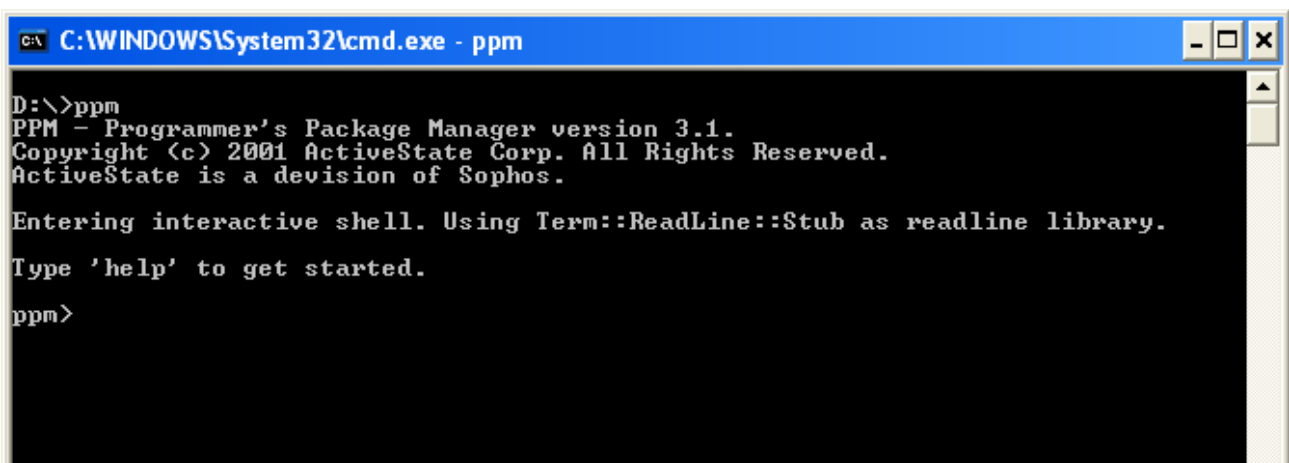
Nel CPAN i moduli Perl sono organizzati in 21 categorie ed ognuna è collegata agli autori che vi hanno contribuito con i propri moduli.

### 13.3 Differenze con i Moduli Standard

Dal punto di vista concettuale non esiste differenza in quanto si tratta sempre di files .pm creati in Perl (magari utilizzando la programmazione ad oggetti e le Espressioni Regolari) è solo che non vengono integrati direttamente con quelle che vengono definite le Librerie di Sistema.

### 13.4 Installazione del CPAN sotto Windows

Se avete installato il Perl dell'ActiveState avete a disposizione un'utility a linea di comando denominata PPM (Perl Package Manager) che vi permette la gestione dei moduli CPAN.



Comando PPM	Descrizione
help	Lista dei comandi accettati dal PPM
install package	Permette l'installazione di un determinato modulo
search package	Verifica che il nome del modulo esista all'interno del repository
remove package	Disinstalla un modulo



rep	Gestione del repository (help rep)
quit	Esce dal PPM

I moduli scaricati ed installati col PPM non vengono direttamente dal repository CPAN ma da quello dell'ActiveState e questo perché la maggior parte di questi nasce per piattaforme Unix e quindi potrebbero non essere completamente compatibili con Windows.

Per averne una lista completa potete far visita al sito ufficiale:

[aspn.activestate.com/ASPN/Modules/](http://aspn.activestate.com/ASPN/Modules/)

### 13.5 Installazione del CPAN sotto Linux

Ci sono due modi per installare i moduli sotto Linux:

- Manualmente
- Modulo CPAN

Manualmente significa scaricarsi il tarball (un file con estensione tar.gz) scompattarlo e da linea di comando lanciare:

```
perl Makefile.PL
make
make test
make install
```

Attraverso il Modulo CPAN invece significa usare una sorta di PPM dell'ActiveState:

```
perl -MCPAN -eshell
```

La prima volta che lo lanciate vi chiede una serie di informazioni (tra le quali anche quale area geografica volete usare per scaricarvi i moduli) per la configurazione che poi si salva su di un file Config.pm

```
cpan>
```

Comando CPAN	Descrizione
?	Lista dei comandi accettati dal CPAN
install package	Permette l'installazione di un determinato modulo
d /regexpr/	Cerca all'interno delle distribuzioni una descrizione che soddisfi la regexpr
a /regexpr/	Come d ma lo cerca all'interno degli autori
quit	Esce dal CPAN

Quest'ultimo sistema è comodo anche per il mantenimento delle dipendenze, infatti se per funzionare un modulo ne utilizza degli altri è necessario che anche quest'ultimi vengano installati; nel primo caso l'installazione abortisce segnalando la mancanza di un certo modulo, con questo invece vengono installati tutti i moduli necessari.

### 13.6 In quale directory si trovano?

---

La variabile speciale `@INC` contiene l'elenco delle directory utilizzate dal Perl per le proprie librerie di file. Con questo semplice script potete avere la lista completa di tutti i moduli presenti sul vostro computer.

```
use File::Find;

my @file;
find \&filePM,@INC;
print join "\n", @file;

sub filePM{
    if (-f $_ and /\.pm$/) {
        push @file, $File::Find::name;
    }
}
```

# 14

# I Database Parte I Introduzione

## 14.1 Quando il gioco si fa duro...

Gli array e gli hash sono strutture per gestire i dati molto usate ed estremamente utili al programmatore, ma quando il volume dei dati da gestire comincia ad essere consistente allora occorre rivolgersi a degli strumenti che fanno di mestiere la gestione affidabile e sicura dei dati: i Database.

## 14.2 Quale Database?

Questo è un Corso di Perl e non mi metterò di certo a stilare pagine di lodi a favore di un database piuttosto che ad un altro, vi riporto la situazione così com'è, ovvero di un semplice elenco di prodotti (opensource e commerciali) che il mercato propone. Si tratta di RDBMS, ovvero DataBase Relazionali (Relation Data Base Management System) che utilizzano come architettura una relazione matematica tra le righe (i records) e le colonne (i campi).

Prodotto	Licenza	Piattaforma	Linguaggio	Elementi	Transazioni	Perl
Oracle 10g	Commerciale*	Win32; Linux; Solaris; Aix	SQL99	Store Procedure, Triggers, Viste	SI	SI
MySQL 4.1	GPL e Commerciale	Win32; Linux	SQL92	Viste	SI con InnoDB	SI
Firebird 1.5	GPL	Win32; Linux; Solaris; OSX; FreeBSD	SQL92	Store Procedure, Triggers, Viste	SI	SI
PostgreSQL 8.0	GPL	Linux; Win32	SQL92	Store Procedure, Triggers,	SI	SI

				Viste		
Microsoft SQL 2003	Commerciale	Win32	SQL99	Store Procedure, Triggers, Viste	SI	SI
SQLite 2	GPL	Linux/Win32	SQL92	Triggers, Viste	SI	SI

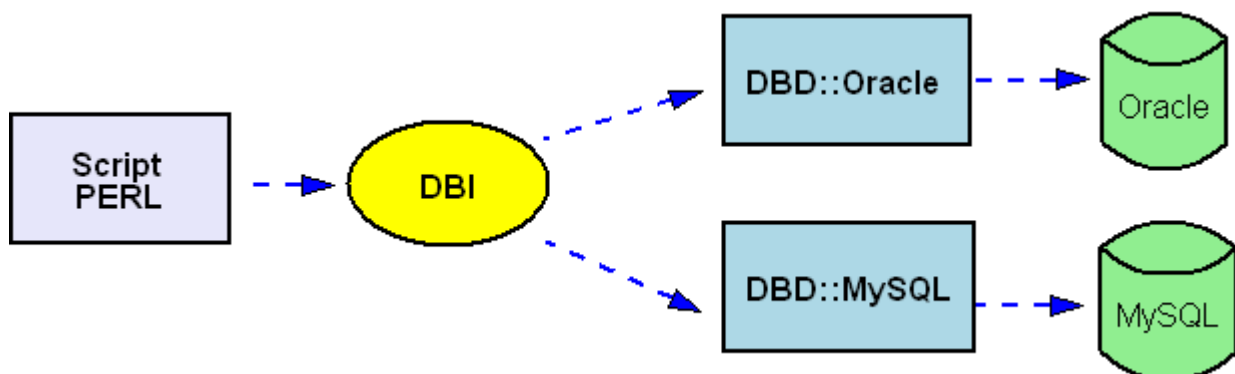
GPL è la Gnu Public Licence, la licenza OpenSource per antonomasia, quando, come nel caso di MySQL è abbinata ad una licenza commerciale significa che è GPL finché lo è anche il software prodotto, dal momento in cui diventa commerciale bisogna pagare la licenza. Nel caso di Oracle invece esiste una sorta di Free at Home in cui un programmatore è libero di scaricarsi ed installare tutti i programmi che vuole finché non decide che è giunto il momento di "uscire" e vendere ciò che ha prodotto, ed allora scatta la licenza commerciale (che può essere accollata al cliente).

### 14.3 Database No Problem

Come avete notato nella tabella, tutti prodotti menzionati hanno delle librerie per interfacciarsi al Perl senza bisogno di strumenti esterni come ODBC o JDBC. Questo perché lo strumento per interfacciarsi al DB il Perl ce l'ha già in casa ed è il DBI (Data Base Interface).

### 14.4 Il Modello DBI

Il Data Base Interface è un'architettura che si divide in due principali gruppi di software; il primo è il DBI stesso mentre il secondo è il DBD (Data Base Driver).



[Figura 1]

Il DBI è un Modulo Indipendente che non si preoccupa del motore di database su cui deve lavorare perché quest'ultimo viene gestito dal driver nativo (DBD) detto anche Modulo Dipendente. E' piuttosto frequente trovare nei manuali la dicitura Data Base Indipendent per DBI e Data Base Dipendent per DBD. Ma questo è insito nello spirito bizzarro del Perl stesso.

Il DBI definisce tre tipi principali di oggetti (handle) per interagire col DB:

- **Handle del Driver**, solitamente non si usa all'interno dei programmi se non per avere la lista dei DBD disponibili (ovvero quelli che abbiamo precedentemente installato)

```
$drh = DBI->available_drivers();
```

- **Handle del Database**, quando si effettua la connessione al DB

```
$dbh = DBI->connect($DSN,$user,$passwd);
```

Il \$DSN viene rappresentato nella forma: `dbi:DriverName:Istanza`  
 Dbi è fisso, DriverName è il nome del driver (oracle, mysql, ecc.)  
 l'Istanza dipende dal DBMS stesso ed indica il DataBase vero e proprio.

- **Handle delle Istruzioni (o Statement)**, quando si esegue un'interrogazione (query) sul DB utilizzando le istruzioni del proprio RDMBS (solitamente lo Standard SQL)

```
$sth = $dbh->prepare("SELECT * FROM TABELLA");
```

Il modulo DBI è orientato agli oggetti, infatti le `->` indicano la relazione tra la Classe (una sorta di funzione) DBI ed il suo metodo (la sotto-funzione) che viene richiamato che può essere: `connect`, `prepare`, `execute`, `disconnect`, ecc.

## 14.5 SQL: Il Linguaggio Universale

Lo Structured Query Language è stato sviluppato da IBM negli anni '70 ed è diventato il linguaggio standard per l'interrogazione dei moderni RDBMS (Relation Data Base Management System). La sua peculiarità è quella mantenere uno strato separato dal motore DBMS vero e proprio permettendo all'Amministratore di gestire il DB usando dei comandi standard.

Per usare un parallelismo è lo Unix dei DataBase.

Nel corso degli anni l'SQL ha subito profonde evoluzioni e numerosi dialetti (ogni produttore aggiungeva funzionalità non standard ai propri RDBMS) finchè nel 1986 l'American National Standards Institute (ANSI) ha deciso per uno standard unico che poi si è evoluto nel corso degli anni. Per questo tra le

specifiche dei prodotti, quando si parla di linguaggio, oltre ad indicare SQL si indica anche lo standard ANSI supportato. Per ora ce ne sono tre:

- SQL detto anche SQL89
- SQL2 detto anche SQL92
- SQL3 detto anche SQL99

Conoscere l'SQL significa poter passare tra prodotti molto diversi come Oracle e MySQL ma utilizzare il medesimo set di istruzioni.

### 14.6 Installazione

---

Per installare il modulo DBI che sia con PPM (Win32) o modulo CPAN (Linux):

```
install DBI
```

Per i driver del DB invece ecco una breve lista di quelli a disposizione:

#### Moduli Driver DBI

Prodotto	Driver
Oracle	DBD::Oracle
MySQL	<b><u>DBD::MySQL</u></b>
Firebird	<b><u>DBD::Interbase</u></b>
Interbase	<b><u>DBD::Interbase</u></b>
PostgreSQL	<b><u>DBD::pg</u></b>
Microsoft ADO	<b><u>DBD::ADO</u></b>
Xbase	<b><u>DBD::Xbase</u></b>
ODBC	<b><u>DBD::ODBC</u></b>
Informix	<b><u>DBD::Informix</u></b>

Una lista più completa la potete trovare su:

[http://search.cpan.org/modlist/Database\\_Interfaces/DBD](http://search.cpan.org/modlist/Database_Interfaces/DBD)

# 15

## I Database Parte II Oracle

### 15.1 Una Scelta Obbligata

---

In un grafico a torta, contenente le percentuali di segmentazione del mercato dei DataBase, l'intera metà della fetta risulta ad appannaggio di Oracle. Questo significa che se siete dei Programmatori o DBA (DataBase Administrator) non potete escludere dalle vostre conoscenze un tale colosso informatico. Persino Microsoft usa Oracle per le proprie applicazioni Mission Critical, eppure anche loro possiedono il loro bel Microsoft SQL Server.

Diciamo che quando performance, sicurezza e robustezza diventano gli unici parametri per misurare l'affidabilità di un sistema, Oracle fa al caso vostro, soprattutto se su piattaforma Linux e a dirlo non è Linus Torvalds ma Oracle stessa.

### 15.2 Installazione

---

Per potersi connettere ad un Server Oracle (che si presume funzionante) da un Client attraverso degli script Perl sono necessarie tre componenti:

- DBI
- DBD::Oracle
- OCI (Oracle Call Interface)

Il DBI l'abbiamo visto nel capitolo precedente, l'OCI invece è un insieme di librerie incluse nel Client Oracle che permettono la connessione al DB dal "mondo esterno" attraverso strumenti come il DBI, appunto.

Per interfacciarsi all'OCI però il DBI ha bisogno del driver nativo, ovvero il DBD::Oracle

L'OCI possiamo recuperarlo attraverso il sito ufficiale (è necessario però prima crearsi un account) all'indirizzo (se non lo cambiano):

[www.oracle.com/technology/software/products/database/oracle10g/index.html](http://www.oracle.com/technology/software/products/database/oracle10g/index.html)

Se poi sulla vostra macchina avete già installato il client Oracle per connettervi attraverso SQLplus al DB Oracle, l'OCI l'avete già incluso.

Per il DBD::Oracle abbiamo due possibilità:

- Installazione da Linux (CPAN)  
`cpan>install DBD::Oracle`
- Installazione da Win32 (PPM ActiveState)  
`PPM>install DBD-oracle`

Dalla Versione 5.8 ActiveState ha tolto dal proprio repository, su pressioni di Oracle, l'omonimo DBD (anche se non si è capito il motivo, chissà) per cui è impossibile installarlo. Cercando su Google però ho trovato un altro repository <http://ftp.esoftmatic.com/outgoing/DBI/> che contiene il modulo `ppd` da installare. Per cui da linea di comando MsDos potete digitare:

```
ppm install ftp://ftp.esoftmatic.com/outgoing/DBI/5.8.3/DBD-Oracle.ppd
```

Mi raccomando questo è valido solo dalla versione 5.8.x (se lanciate `perl -v` vedete la versione che avete installato).

### 15.3 Connessione al DB

La prima istruzione è il connect al DB

```
$dbh=DBI->connect(
    "dbi:Oracle:tnsconnect",
    "user",
    "passwd",
    $options) || die "$DBI::errstr";
```

- `$dbh` è l'handle del DB
- `tnsconnect` è l'istanza del DB
- `$options` non è obbligatorio e può contenere tre parametri: `RaiseError`, e `PrintError` agiscono sulla variabile `$DBI::errstr` per il semplice warning o l'indicazione dell'errore bloccante; `AutoCommit` invece se è a 1 forza il commit del DB ad ogni transazione riuscita anziché alla disconnessione del DB come di default.

### 15.4 Query SQL

Una volta connesso, il DB può essere interrogato usando l'istruzione SQL più comune, la SELECT.

```
$stmt = $dbh->prepare("select * from TABELLA") || die DBI::errstr;
$rc = $stmt->execute() or die $DBI::errstr;
```



Il `prepare()` verifica la sintassi SQL (parser) mentre l' `execute()` la esegue. Questa pratica però tende ad essere inefficiente nel momento in cui le query sono ripetitive, per questo esiste il metodo `prepare_cached()` che si preoccupa di verificare che l'istruzione precedente sia diversa da quella in corso.

Per recuperare i risultati della query ci sono diversi metodi (ricordo che si tratta di moduli object-oriented) uno dei più usati è `fetchrow()` che riempie un array come fossero i campi di un record.

```
while((@myrec) = $stmt->fetchrow()) {
    $ncampo = 1;
    foreach $field (@myrec){
        print "\nCampo [$ncampo] = $field";
        $ncampo++;
    }
}
print "\nTotale Records " . $stmt->rows;
```

Se però conosciamo già la struttura della nostra Tabella possiamo usare `bind_columns()` per definirla e `fetch()` per elencarne il contenuto.

```
$stmt->bind_columns(undef, \$id, \$titolo, \$autore, \$argomento);
while($stmt->fetch()) {
    print "\n$id, $titolo, $autore, $argomento";
}
print "\nTotale Records " . $stmt->rows;
```

Quando invece vogliamo usare comandi che agiscono sulla struttura del DB, come la creazione di tabelle o la loro cancellazione, usiamo il metodo `do()`

```
$sql = <<EOF_SQL
    CREATE TABLE impiegati (
        id INTEGER NOT NULL,
        nome VARCHAR2(128),
        titolo VARCHAR2(20),
        telefono VARCHAR2(15)
EOF_SQL
;
$dbh->do($sql);
```

## 15.5 INSERT e UPDATE

Se vogliamo inserire nuovi records faremo la `prepare()` del relativo comando SQL (`INSERT INTO`) aggiungendo dei punti interrogativi nei campi che andranno compilati al momento della `execute()`

```
$stmt = $dbh->prepare("INSERT INTO LIBRI VALUES (?, ?, ?, ?)");
$rc = $stmt->execute(4, "ORACLE for DBA", "TIGER SCOTT", "ORACLE");
```

Stessa cosa per l'aggiornamento (UPDATE) di un record esistente

```
$sql = "UPDATE LIBRI SET AUTORE=? WHERE ID_LIBRO = 1";
$stmt = $dbh->prepare($sql);
$rc = $stmt->execute("TIM BOUNCE & ALLIGATOR DESCARTES");
```

Molti anzichè usare il punto interrogativo inseriscono i campi come scalari all'interno della stessa `prepare()` in questo modo:

```
$sql = "INSERT INTO LIBRI VALUES ($id,$titolo,$autore,$genere)";
```

Questo può mandare in abort l'intera Query nel caso in cui all'interno degli scalari vi fossero degli apici

Il metodo `execute()` non aggiorna immediatamente il DB a questo ci pensa il `commit` che necessita dell'handle creato dal DBI

```
$dbh->commit;
```

## 15.6 Le Transazioni

Per Transazione si intende un sequenza indivisibile (atomica) di comandi sul DB tale che o ha un successo totale oppure viene annullata completamente. Il classico esempio è quello delle Transazioni Bancarie in cui ad un prelievo devono seguire tutta una serie di operazioni che devono andare a buon fine, pena la squadratura dei Conti. Vedremo quindi che a seguito di una `prepare()` ci sarà un blocco di codice delimitato da `eval{ }` al cui interno vi saranno una serie di `execute()` e relative `commit()`. Se si verifica anche un solo errore verrà innescata un `rollback()` per ritornare alla situazione precedente al blocco `eval{ }`

```
@rek=(
    [1, "PERL", "LARRY WALL", "PERL"],
    [2, "PERL E DBI", "BRUCE", "PERL"],
    [3, "ORACLE for DBA", "TIGER SCOTT", "ORACLE"]);

$stmt = $dbh->prepare("INSERT INTO LIBRI VALUES (?, ?, ?, ?)");

for (@rek) {
    eval {
        $stmt->execute($rek);
        $dbh->commit();
    }
}
```

```

};
if ($@) {
    print "Transazione Abortita \n$DBI::errstr\n";
    $dbh->rollback();
}
}

```

### 15.7 Le Stored Procedures

---

Una Stored Procedure è una funzione creata all'interno del DataBase e richiamabile come fosse una subroutine; il DBI non ne prevede una gestione standard, ma possiamo ovviare col `do()` nella forma:

```
$dbh->do("BEGIN myPackage.myProcedure; END;");
```

### 15.8 Disconnessione

---

Quando terminiamo la connessione con

```
$dbh->disconnect() || die "Errore Logoff Server Oracle";
```

mandiamo in `commit` tutte le operazioni mandate in `execute()`

# 16

## I Database Parte II MySQL

### 16.1 L'Alternativa Open

---

Se Oracle rappresenta la scelta obbligata per quanto riguarda l'adozione di un software commerciale di alto livello, MySQL incarna l'alternativa Open più per l'ampia comunità di sviluppatori che ha abbracciato piuttosto che per le reali specifiche di similitudine con Oracle stesso. Nel panorama del software libero esistono altre realtà come PostgreSQL e Firebird (ex-Interbase di Borland) che maggiormente possono avere dei punti di contatto con quella architettura di RDBMS. MySQL è un DB molto flessibile che grazie all'escalation della configurazione LAMP (Linux, Apache, MySQL e PHP) è diventato uno standard de-facto nel mondo dell'Information Technology.

### 16.2 Quale Versione?

---

Personalmente utilizzo la 4.1 su un paio di Server ed anche su Internet parecchi servizi di hosting lo hanno già adottato, ed è su questa versione che ho fatto tutte le prove degli script. E' piuttosto stabile, certo non avrà Store Procedure e Triggers come la 5.0 ma ha dalla sua parte un ben più lungo periodo di testing. Considerando soprattutto il fatto che al momento della stesura di questo Corso, la 5.0 viene ancora definita BETA.

### 16.3 Installazione

---

Anche per MySQL sono necessarie tre componenti:

- DBI
- DBD::mysql
- MySQL Client

Il DBI abbiamo visto come installarlo mentre il Client MySQL lo potete trovare su [dev.mysql.com/downloads/](http://dev.mysql.com/downloads/)

Per driver DBD::mysql invece avete le solite due scelte:

- Installazione da Linux (CPAN)  
# perl -MCPAN -e "install DBD::MySQL"
- Installazione da Win32 (PPM ActiveState)  
c:\ppm install DBD-mysql

### 16.4 Connessione al DB

Rispetto ad Oracle con MySQL è più facile avere una situazione in cui il Client ed il Server sono la stessa macchina per cui quando si vuole specificare l'Hostname si indica localhost (è l'indirizzo di loopback 127.0.0.1 che puntano tutte le macchine)

```
$hostname = "localhost";
$port = "";
$database="test";
$user = "alex";
$password = "mysql1";

$dbh = DBI->connect(
    "dbi:mysql:database=$database;host=$hostname;port=$port",
    $user, $password) || die "$DBI::errstr";
```

- \$dbh è l'handle del DB
- \$port è la porta tcp lasciata in ascolto dal server, se rimane vuota viene utilizzata quella di default

### 16.5 Query SQL

Come nell'esempio in Oracle anche in questo caso la Query avviene prima con la prepare() del comando SQL e poi con l'execute(). Sfortunatamente non esiste una prepare() che sfrutta la cache dei comandi SQL precedentemente lanciati come per Oracle.

```
$sth = $dbh->prepare("SELECT * FROM TABLE") || die $DBI::errstr;
$rc = $sth->execute() || die $DBI::errstr;
```

Mentre per recuperare i record abbiamo una vera e propria famiglia di fetchrow\_()

fetchrow_array	Metodo classico di fetchrow() a cui associa un campo del record ad ogni elemento dell'array
fetchrow_arrayref	Anziché un array ritorna un

	Reference dell'array
fetchrow_hashref	Questo è il più interessante in quanto col Reference di un hash lo possiamo associare al nome del campo anziché al suo numero progressivo
fetchall_arrayref	Mette l'intera Query in un array bidimensionale

Col metodo standard `fetchrow_array()` abbiamo in tutto e per tutto `fetchrow()` tanto che lo potete chiamare nella stessa maniera

```
while((@myrec) = $stmt->fetchrow_array()){
    print "\n";
    $ind=0;
    foreach $myrec (@myrec){
        print "Campo($ind)=[ " . $myrec . " ] ";
        $ind++;
    }
}
print "\nTotale Records " . $stmt->rows;
```

Col metodo `fetchrow_hashref()` invece bisogna conoscere il nome dei campi

```
while ($hash_ref = $stmt->fetchrow_hashref){
    print "\n TITOLO = " . $hash_ref->{TITOLO};
    print "\n AUTORE = " . $hash_ref->{AUTORE};
}
```

Infine con `fetchall_arrayref()` riempiamo un array non con un record alla volta ma con l'intera Query (quindi attenzione a quello che fate). Avremo quindi una matrice vera e propria il cui numero di righe (i records) corrisponde a `#{ $table }` ed il numero di colonne (i campi) a `#{ $table->[ $irek ] }`

```
$table = $stmt->fetchall_arrayref;
for($irek=0;$i<=#{ $table };$i++) {
    for ($icol=0;$icol<=#{ $table->[ $irek ] };$icol++){
        print "\n$table->[ $irek ][ $icol ]\t";
    }
    print "\n";
}
```

Qualche altro Metodo interessante (valido anche per Oracle) abbiamo `{ NUM_OF_FIELDS }` per sapere quanti campi ha ogni record e `rows()` quanti records sono stati conteggiati dalla Query (che sia una semplice `SELECT` oppure un `INSERT INTO` o anche un `DELETE`)

Mentre più specifico per MySQL abbiamo `{insertid}` comodo se usiamo un campo `AUTO_INCREMENT` e vogliamo sapere che progressivo gli verrà assegnato.

```
$next_id = $stmt->{insertid};
```

## 16.6 INSERT e UPDATE

---

Valgono le stesse cose dette per Oracle, ovvero `prepare()` del relativo comando SQL (`INSERT INTO`) ed aggiunta dei punti interrogativi nei campi che andranno compilati al momento della `execute()`

```
$stmt = $dbh->prepare("INSERT INTO LIBRI VALUES (?, ?, ?, ?)");
$rc = $stmt->execute(5, "MYSQL", "SAVERIO RUBINI", "MYSQL");
```

Stessa cosa per l'aggiornamento (`UPDATE`) di un record esistente

```
$sql = "UPDATE LIBRI SET TITOLO=? WHERE ID_LIBRO = 5";
$stmt = $dbh->prepare($sql);
$rc = $stmt->execute("MYSQL POCKET");
```

## 16.7 Disconnessione

---

Come spiegato in precedenza

```
$dbh->disconnect() || die "Errore Logoff Server MySQL";
```

## 16.8 Ma Le Istruzioni Sono Sempre le Stesse?

---

Beh, diciamo che uno dei motivi per cui l'architettura DBI ha preso vita erano quelli di avere un set comune di istruzioni Indipendenti dal DB a cui applicarle. Come potete notare i Metodi DBI tendono ad avere il medesimo nome - ad esempio sia in Oracle che in MySQL c'è `fetchrow()` ma in quest'ultimo esistono delle varianti come `fetchrow_array()` - o al limite esistono delle implementazioni da una parte e non da un'altra perché sono correlate all'architettura dell'RDMBS stesso.

Negli ultimi due Capitoli verranno proposti degli esempi che implementano dei DB in MySQL.

# 17

## Win32

### 17.1 Aprite le Finestre!

Gli aspiranti programmatori Perl che sanno già che non toccheranno nemmeno con un mouse una finestra di Windows perché sono totalmente presi dai loro sistemi Unix-Like possono saltare a piè pari questo capitolo, ma per tutti gli altri ecco quello che stavate aspettando da tempo. Un completo set di istruzioni Perl fatte apposte per la vostra piattaforma preferita; Windows NT!

E' uso comune riferirsi a Win32 per intendere tutte le piattaforme Windows di Microsoft dal 95 al 2003, per Perl non è così per lui Win32 sono tutte le piattaforme avanzata (non nel senso che sono rimaste invendute) da NT4 per intenderci, fino al Server 2003. Windows 2000 ed XP si trovano nel mezzo.

### 17.2 use Win32

Win32 non è un singolo modulo da cui emergono una serie di funzioni bensì un'intera famiglia inclusa nei famosi Moduli Standard del Perl e richiamabile dall'istruzione:

```
use Win32;
```

Modulo	Descrizione
Win32::Clipboard	Interazione con la Clipboard (Appunti) di Windows
Win32::Console	Interazione con la Console di Windows
Win32::ChangeNotification	Crea e utilizza gli oggetti Change/Notification
Win32::EventLog	Legge e scrive nel log degli eventi di NT
Win32::File	Gestisce gli attributi di file (attrib)
Win32::FileSecurity	Gestisce gli ACL in Perl
Win32::Internet	Fornisce le estensioni per i servizi internet
Win32::IPC	Attende oggetti (processi, mutex, semafori)
Win32::Mutex	Crea e utilizza I mutex
Win32::NetAdmin	Amministra utenti e gruppi



Win32::NetResource	Gestisce le risorse (server, condivisione di file, stampanti)
Win32::Process	Avvia e interrompe i processi Win32
Win32::Registry	Legge e gestisce il Registry di Win32
Win32::Semaphore	Crea e usa semafori
Win32::Service	Gestisce I servizi di Windows NT
Win32::Shortcut	Interfaccia con il collegamento shell

I moduli come vedete sono parecchi (senza contare quelli all'interno del CPAN) ed analizzarli tutti servirebbe un altro Corso, ho privilegiato quindi quelli che non potevano essere surrogati da quelli fino ad ora descritti e che avessero un'attinenza con le esigenze più comuni.

### 17.3 use Win32::Registry

E' probabilmente il modulo più interessante dell'intero pacchetto poiché fornisce l'accesso diretto al Registro di Windows, il database che memorizza le informazioni relative a tutti i componenti del sistema e del software.

Lavorare col Registry significa avere ben chiaro il concetto "Finchè lavoro in lettura non c'è problema", se invece decidiamo di voler modificare qualcosa allora facciamo tutti i backup possibili ed immaginabili e teniamoci forte. Il sottoscritto comunque declina ogni responsabilità sull'utilizzo degli esempi proposti.

Il Registry di Windows prevede cinque chiavi principali (HKEY\_) ognuna delle quali con una funzionalità ben precisa.

- \$HKEY\_CLASSES\_ROOT
- \$HKEY\_CURRENT\_USER
- \$HKEY\_LOCAL\_MACHINE
- \$HKEY\_USERS
- \$HKEY\_CURRENT\_CONFIG

```
HKEY->Open($subKey, $myhKey);
```

Aprirò il registro come se fosse un file posizionandomi direttamente alla chiave indicata con \$subKey ed ottenendo l'Handle \$myhkey

```
$subKey= "Software\\Microsoft";
$HKEY_LOCAL_MACHINE->Open($subKey,$myhkey) || die "Errore!";
```

Una volta aperto il Registro ecco una lista di tutte le operazioni che posso effettuare.

Create(\$subkey, \$newkey)	Crea una nuova chiave di registro
----------------------------	-----------------------------------

SetValue(\$subKey,\$tipo,\$value)	Imposta un valore in \$subkey
QueryValue(\$subkey,\$value)	Restituisce il valore della \$subkey in \$value
QueryKey(\$key,\$subkey,\$vals)	Ritorna la \$key di Registro in base al alla sotto-chiave e al suo valore
GetKeys(\@array)	Restituisce un @array contenente tutte le sotto-chiavi
GetValues(\%hash)	Restituisce un %hash contenente chiave, tipo valore e valore
Save(\$FileName)	Salva su di un \$FileName la radice dell'albero del Registry
Load(\$subkey,\$FileName)	Carica un file Registry nella sottochiave \$subKey specificata
DeleteKey(\$subkey)	Elimina dal Registry la \$subkey specificata
DeleteValue(\$subkey)	Elimina dalla \$subkey il suo valore
Close()	Chiude la sessione aperta con Open

A questo punto posso ottenere la lista di tutte le sotto-chiavi ed andarle a mettere in un array @key\_list (occhio al reference).

```
$myhkey->GetKeys(\@key_list);
```

In questo modo ottengo l'elenco di tutte le Chiavi poste al livello di \$subkey

```
foreach $key (@keys) {
    print "\nChiave = $key";
}
```

Se invece voglio ottenere sia il nome della Chiave che il rispettivo Valore devo usare una funzione che utilizzi un hash anziché un'array

```
$myhkey->GetValues(\%kvalue);
foreach $value (keys(%kvalues)) {
    $RegType = $kvalues{$value}->[1];
    $RegValue = $kvalues{$value}->[2];
    $RegKey = $kvalues{$value}->[0];
    print "\n[$RegKey] => $RegValue";
}
```

Prima di far danni magari mi faccio una copia di backup del Registro (come se bastasse ad evitare i guai)

```
$myhkey->Save("c:\registro_bak.reg");
```

Se vogliamo creare una nuova chiave di registro

```
$hkey->Create("NuovaChiave", $SubKey);
```

e per finire...

```
$myhkey->Close();
```

Concludo dicendo che Win32::Registry è compatibile anche con Windows 9x così possiamo rovinare sia il 95 e che il 98 (Windows ME si è rovinato da solo).

### 17.4 use Win32::Console

Questo modulo implementa la Console di Win32 e le funzioni di modalità carattere. Queste forniscono il controllo completo dell'Input/Output di Console, ossia lettura e scrittura di caratteri, attributi e di intere porzioni dello schermo.

Per prima cosa creiamo l'oggetto Console ed usiamo il suo handle (un po' come accade con l'apertura di un file) come riferimento per innescare i suoi Metodi.

```
$chandle = Win32::Console->new(Tipo_handle);
```

Tipo\_handle:

- STD\_OUTPUT\_HANDLE
- STD\_INPUT\_HANDLE
- STD\_ERROR\_HANDLE

```
$CONSOLE = Win32::Console->new(STD_OUTPUT_HANDLE);
```

A questo punto possiamo usare le funzioni CIs() e WriteChar() per pulire lo schermo, scrivere una stringa e posizionarla in una zona dello schermo.

```
$CONSOLE->Cls();
$CONSOLE->WriteChar("Prova di Stampa Riga 1 Colonna 0",0,1);
$CONSOLE->WriteChar("Prova di Stampa Riga 5 Colonna 0",0,5);
```

con Info() abbiamo un array con lo status della Console

```
@array = $CONSOLE->Info();

[0..1]   colonna,riga buffer di console
[2]      colonna corrente
[3]      riga corrente
[4]      attributo corrente per Write()
[5]      colonna di sinistra (left)
[6]      riga superiore (top)
[7]      colonna di destra (right)
[8]      riga inferiore (bottom)
[9]      numero massimo di colonne
```

```
[10]         numero massimo di righe
print "Cursore alla riga $array[3] e colonna $array[2]";

con Free() liberiamo l'oggetto $CONSOLE istanziato dal Console->new()
$CONSOLE->Free();
```

### 17.5 use Win32::File

---

Permette di vedere o impostare gli attributi dei file.

```
Win32::File::GetAttributes($FileName,$attr);
Win32::File::SetAttributes($FileName,$attr);
```

Gli attributi sono quelli classici di Windows, ovvero:

- ARCHIVE
- COMPRESSED
- DIRECTORY
- HIDDEN
- NORMAL
- OFFLINE
- READONLY
- SYSTEM
- TEMPORARY

\$attr contiene la somma logica di uno o più dei valori di attributo sopra indicati

```
if ($attr & DIRECTORY) {
    print "<DIR>";
}
```

### 17.6 use Win32::Service

---

Permette la gestione dei servizi su macchine NT (NT4,2000,XP,2003)

Per cominciare può essere utile conoscere la lista di tutti i servizi presenti sulla nostra macchina:

```
Win32::Services::GetService($hostname,%hash);
```

In questo caso riempiamo l'hash (passato come reference alla funzione GetService) con la descrizione del servizio come chiave ed il nome del servizio come valore. L' \$hostname (se lasciato "" usa localhost) è il nome Netbios della macchina oppure il suo indirizzo IP.

Di seguito i comandi la cui spiegazione sul funzionamento è insita nel nome:

```
Win32::Services::StartService($hostname,$servizio);
Win32::Services::StopService($hostname,$servizio);
Win32::Services::PauseService($hostname,$servizio);
Win32::Services::ResumeService($hostname,$servizio);
Win32::Services::GetStatus($hostname,$servizio,\%status);
```

GetStatus() visualizza lo status di un servizio in forma di hash le cui chiavi sono:

ServiceType, CurrentState, ControlsAccepted, Win32ExitCode, ServiceSpecificExitCode, CheckPoint, WaitHint.

### 17.7 Estensioni Win32

Elenco di seguito una serie di estensioni (o funzioni) che il modulo Win32 include automaticamente. Hanno tutte il suffisso Win32:::

GetLastError()	Restituisce l'ultimo errore generato
LoginName()	Nome dell'utente che ha lanciato lo script
DomainName()	Nome del Dominio NT
FsType()	Tipo di FileSystem dell'unità corrente
GetCurrentDir()	Percorso corrente
SetCurrentDir(\$dir)	Imposta la Directory corrente
GetOSVersion()	Restituisce l'esatta release di Windows in formato array
IsWinNT()	Restituisce true se è un sottosistema NT
IsWin95()	Restituisce true se è un sottosistema 9x
GetShortPathName(\$dir)	Restituisce il percorso in formato corto (8+3)

Ecco un modo per avere il dettaglio della versione di Windows:

```
($stringa, $major, $minor, $build, $id)=Win32::GetOSVersion();
```

### 17.8 In Conclusione

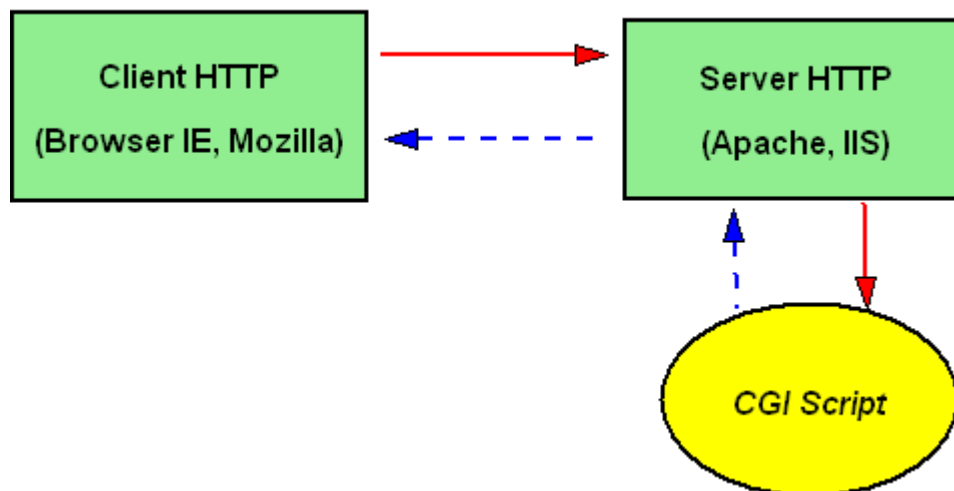
Esistono parecchi moduli fatti ad arte per la piattaforma Windows; a mio avviso andrebbero usati solo quelli che sono realmente necessari poichè alcuni moduli fatti per Unix sono completamente inutilizzabili per Win32 ma altri sono complementari e quindi farne un largo uso compromette la possibilità di far girare il medesimo script su altri sistemi. E' chiaro che se la priorità è lavorare su server Windows allora questi moduli sono necessari.

# 18

## CGI ed Internet

### 18.1 Quei Favolosi Anni '90

Sembra ormai un'eternità, ma in pieno boom da Internet, la NCSA (da cui nacque il WebServer Apache) inventò lo standard che sta alla base di tutti i programmi eseguibili su web che producono un output in HTML, il CGI. Il Common Gateway Interface è un modello di interazione tra Client e Server per permettere l'esecuzione di programmi remoti.



[Figura 1]

In parole povere il CGI permette a programmi diversi, scritti in linguaggi diversi, di collegarsi e di comunicare tra di loro. Il Perl da questo punto di vista, è rimasto per lungo tempo l'unico linguaggio di scripting multiplatforma che permetteva questo meccanismo.

### 18.2 Il Linguaggio HTML

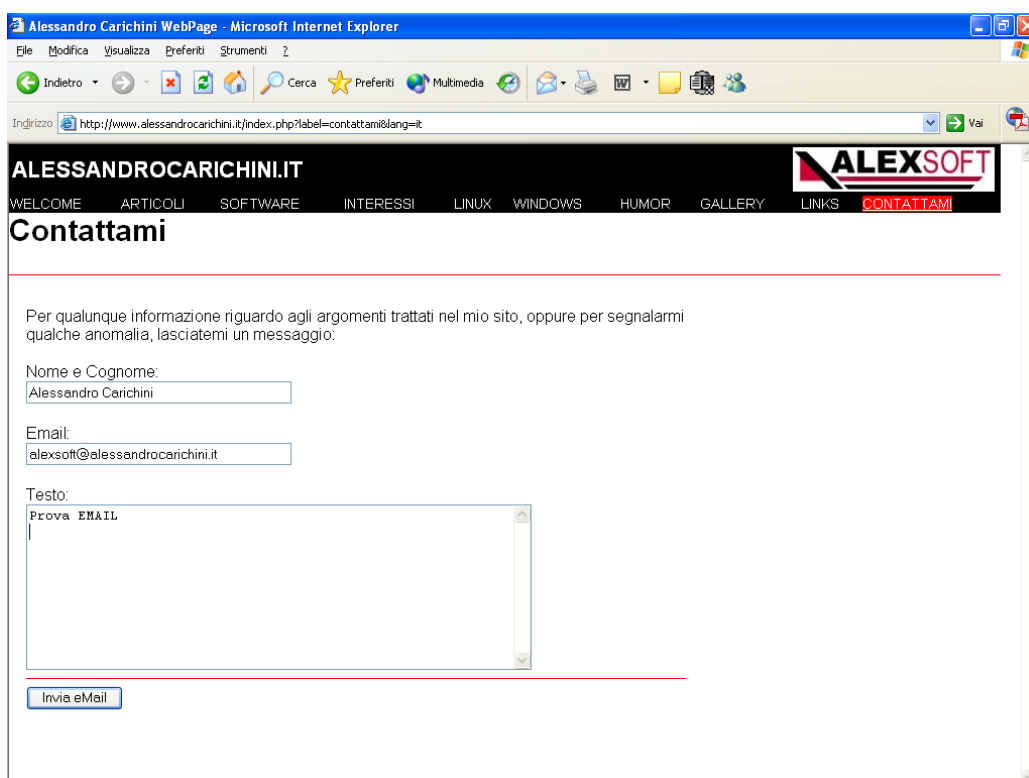
Conoscere le basi del linguaggio HTML diventa un requisito importante anche per programmare in Perl. Infatti se lo standard output di uno script finisce a

video o in un file di testo, lo standard output di un programma CGI va a finire sul web e quindi deve essere formattato in HTML. Diciamo che per cominciare è sufficiente conoscere le basi per la formattazione tipografica del testo perché per il resto ci pensano il set di funzioni incorporate nel modulo CGI.

Tag HTML	Significato
<P>	Doppia Interlinea
 	Singola Interlinea
<B>	Grassetto
<H1>	Intestazione Carattere 1
<H2>	Intestazione Carattere 2

### 18.3 Le Form Web

Questo meccanismo nasce dall'interazione tra l'utente (Browser) e la macchina (Web Server) che avviene attraverso un'interfaccia Web denominata Form. Rappresentano l'ideale sostituzione delle interfacce grafiche (GUI) dei programmi che girano sui computer. Quando ci si trova di fronte ad una tipica Form come questa



[Figura 2]

Possiamo notare che finite le TEXTFIELD (dove si inseriscono i dati) c'è sempre un pulsante per l'invio dei dati appena inseriti. Queste informazioni possono venire spedite in due maniere:

- Metodo GET
- Metodo POST

Col Metodo GET accodiamo le nostre informazioni all'URL della pagina web subito dopo il punto interrogativo nel formato:

```
?Variabile1=Contenuto&Variabile2=Contenuto
```

Col Metodo POST invece le stesse informazioni vengono impacchettate e spedite in maniera nascosta al Web Server senza mostrare alcunchè sull'URL, inoltre non si corre il rischio che il numero di caratteri superi il massimo consentito dalla riga di URL del browser stesso.

### 18.4 use CGI

Cominciamo subito a dire che usando le CGI il Perl viene eseguito sul WebServer per cui ci sono delle piccole accortezze da apporre ai propri script. Lo She-Bang in questo caso è molto importante perché dice al Server dove andarsi a cercare l'interprete Perl, per cui se il sistema operativo è Unix tutto rimane invariato, se invece è Win32 allora bisogna indicargli il percorso esatto.

```
#!c:/perl/bin/perl.exe
```

Un'altra cosa importante è che per funzionare i programmi CGI devono essere posizionati in una certa directory indicata dal WebServer, solitamente si usa `cgi-bin` ma è solo una pura convenzione.

Iniziamo a vedere il dettaglio dello script generato dalla Figura 2

```
use CGI;
$cgi = CGI->new();
$cgi->header();
```

Queste quattro righe sono fondamentali per non ricevere un errore dal WebServer e rappresentano l'inizializzazione dell'oggetto `$cgi`. Cominciamo ora a costruire la pagina Web vera e propria iniziando dal suo Tag di apertura emulato dalla funzione `start_html()`

```
print $cgi->start_html($TitoloDellaPagina);
```

Leggiamo i parametri passati dalla Form.

```
$nome=$cgi->param('nome');
```



```

$email=$cgi->param('email');
$testo=$cgi->param('testo');
$menu=$cgi->param('menu');

if ($menu == 0) {
    print $cgi->startform('GET',"/cgi-bin/contattami.pl");
    print "<P>Nome e Cognome: <BR>";
    print $cgi->textfield("nome",$nome,50,50);
    print "<P>EMAIL: <BR>";
    print $cgi->textfield("email",$email,50,50);
    print "<P>Testo: <BR>";
    print $cgi->textarea("testo",$testo,10,60);
    print $cgi->hidden("menu",1);
    print "<P>";
    print $cgi->submit("Invia eMail");
}else{
    print "<B>HAI DIGITATO!</B><P>";
    print "<P>Nome e Cognome: $nome";
    print "<P>EMAIL: $email";
    print "<P>Testo: <BT>$testo";
}
    
```

Come potete notare oltre all'uso dei `textfield()` e `textarea()` per l'input dei dati abbiamo usato `hidden()` che risulta molto comodo per il passaggio di parametri ad uso del programmatore, nel nostro caso la variabile `$menu` fungeva da switch per capire se il programma doveva mandare in output la richiesta della Form oppure stampare i risultati della Form stessa.

Terminiamo la pagina con il Tag di chiusura

```
print $cgi->end_html();
```

## 18.5 Perl e CGI per il Futuro?

Forse l'accoppiata Perl e Web non è più di gran moda soprattutto dopo l'avvento di linguaggi ben più semplici come PHP e FLASH ma rappresenta da un lato ancora un valido strumento non solo per interfacciare DataBase ma anche per la costruzione di Intranet Aziendali.

# 19

## Networking

### 19.1 La Rete

---

Il Perl oltre alla manipolazione di stringhe e file sembra tagliato per il Networking, infatti grazie alla grande quantità di moduli previsti è diventato il sostituto ideale per i programmatori C che hanno bisogno di uno strumento affidabile e semplice da gestire.

Quando si parla di Rete in realtà non si intende un unico protocollo universale ma tanti a seconda del servizio che dobbiamo gestire. Si tratta di diversi sotto-moduli facente capo a Net: : \*

Servizio	Descrizione	Modulo CPAN
TELNET	Connessione Remota a Terminali Unix	Net::Telnet
PING	Controlla un Host Remoto	Net::Ping
TIME	Date e Ora da un Host	Net::Time
FTP	Trasmissione File	Net::FTP
SMTP	Invio Posta Elettronica	Net::SMTP
POP3	Ricezione Posta Elettronica	Net::POP3
WWW	Librerie WWW	LWP: : *

Al di fuori di questa lista esiste un modulo con funzione singola molto utile per avere il proprio Hostname

```
Use Sys::Hostname;
$host = hostname();
```

### 19.2 use Net:::Ping

---

E' probabilmente l'utility più usata, non solo dagli Amministratori di Rete ma anche dai cosiddetti Power User quando sono alle prese con la loro LAN casalinga. Il Ping sostanzialmente invia un messaggio (datagramma) ad una macchina remota aspettandosi una risposta.

```
$ping = Net:::Ping->new();
```

```

if ($ping->ping($HostName,$timeout) {
    print "\n$HostName Raggiungibile";
}
$ping->close();

```

L'Hostname è chiaramente l'Indirizzo IP della macchina remota mentre Timeout è il numero di secondi (se omesso di default è 5) che la funzione ping() attende per la risposta. E' una sorta di radar (da cui il nome onomatopeico) che permette di capire se una macchina remota è raggiungibile o meno.

### 19.3 use Net::Telnet

---

Questo modulo permette una connessione con un Server Telnet remoto ed eseguire dei comandi da console.

Innanzitutto il Modulo va installato seguendo le solite indicazioni;

Da Win32

```
ppm install Net-Telnet
```

Da Unix

```
perl -MCPAN -e "install Net::Telnet"
```

Solitamente il Telnet è un servizio presente più sulle macchine UNIX che su quelle Win32 (anche se su XP il servizio di default è solo disattivato) per i cui comandi da lanciare devono essere chiaramente compatibili col sistema operativo ospitante.

Una volta istanziato l'oggetto Telnet col ->new

```

$username = "prova";
$password = "prova";
$hostname = "houston";
$telnet = Net::Telnet->new ( Timeout=>10,Errmode=>'die');
$telnet->open($hostname);

```

Abbiamo due modalità di accesso:

#### **Metodo #1**

```

$telnet->waitfor('/login: $/i');
$telnet->print($username);
$telnet->waitfor('/password: $/i');
$telnet->print($password);
$telnet->waitfor('/\ $ $/i');
@output = $telnet->print('who');
print @output;

```

Questo tipo di interfaccia prevede l'accoppiata `waitfor()` e `print()`; il primo rimane in attesa finchè non si trova una certa stringa mentre il secondo sopraggiunge di seguito per mandare in input un eventuale comando.

### **Metodo #2**

```
$telnet = Net::Telnet->new ( Timeout=>10,Errmode=>'die' );
$telnet->open($hostname);
$telnet->login($username,$password);
@output = $telnet->cmd('who');
print @output;
```

Per molti Server l'accoppiata vincente è `login()` e `cmd()`, ovvero loggati con un account valido e poi inviati dei comandi.

La disconnessione invece è comune:

```
$telnet->close();
```

Per collegarsi ad apparati di Rete come ad esempio i Router della famiglia CISCO, esiste un apposito modulo `Net::Telnet::Cisco`

### **19.4 use Net::Time**

Con questo modulo è possibile ottenere l'orario da una macchina remota (Host). Usando il Time Server dell'Istituto Galileo Ferraris [www.ien.it](http://www.ien.it) siamo sicuri di avere l'ora esatta.

```
$TimeServer = "ntp.ien.it";
$timeStamp = Net::Time::inet_time($TimeServer,'tcp',$timeout);
$myDate = Net::Time::inet_daytime($TimeServer,'tcp',$timeout);
```

con `inet_time()` ottengo l'orario nel formato `TimeStamp`, lo stesso della funzione `time()`, mentre con `inet_daytime()` la stringa che otterrei con la funzione `localtime()` vista in precedenza.

```
Sat Jul 16 11:28:44 2005
```

## 19.5 use Net::FTP

---

Il File Transfert Protocol è appunto un protocollo per la trasmissione dei file tra Client FTP e Server FTP. Il Server deve già esistere mentre il Client ce lo mette il Perl. Non so se vi è mai capitato di usare un Client FTP a linea di comando in cui una volta loggati dovete lanciare i comandi `put` e `get` per inviare o ricevere file. Beh, col Perl il senso rimane invariato, solo che il tutto avviene all'interno di uno script.

### 19.x.1 Connessione al Server

```
$hostFTP = "ftp.nai.com";
$username = "anonymous";
$password = "mionome\@dominio.it";
$ftp = Net::FTP->new($hostFTP);
```

### 19.x.2 Autenticazione

```
$ftp->login($username,$password);
```

### 19.x.3 Download

```
# Mi posiziono sulla directory interessata
$ftp->cwd("/pub/antivirus/datfiles/4.x");
# Indico la natura del file (binario)
$ftp->binary();
# Download del File Sorgente in quello di destinazione
$ftp->get($fileSorgente,$fileDestinazione);
```

### 19.x.4 Disconnessione

```
$ftp->quit;
```

Il `get()` come pure il `put()` si aspettano sia per il sorgente che per la destinazione un nome file esistente, pena abort del comando, per cui l'uso di wildcard come gli asterischi non è ammesso. Per questo ci viene in aiuto una funzione come `ls()` che, come l'omologa Unix, permette di elencare una dir con i giusti parametri.

```
@lsfiles=$ftp->ls("*.zip");
foreach $dlfile (@lsfiles){
    print "\nDownload $dlfile";
    $ftp->get($dlfile,$dlfile) or die "Errore $!";
}
```

In questo esempio ci stiamo scaricando tutti i files .ZIP che sono presenti nella directory remota in cui ci siamo spostati col `cwd()`

In questi casi per la gestione dell'errore anziché il brutale `die` vi consiglio l'utilizzo di uno scalare come `switch`. Ad esempio:

```
$ftp->get($dlfile,$dlfile) or die "Errore $!";
```

Diventa:

```
$newerr=0;
$ftp->get($dlfile,$dlfile) or $newerr=1;
if ($newerr == 1) {
    print "\nFile $dlfile non Copiato!";
}
```

Evita che il singolo errore blocchi l'intero script. E' comunque un esempio che può essere applicato ad altre realtà quando si vuole una gestione non bloccante di un errore.

Un altro modo per verificare il corretto download di un file è quello di verificare la dimensione di partenza con quella di arrivo. Su linee particolarmente lente può accadere di avere un download parziale di un file. La funzione `size()` ci dice quando pesa il file su host mentre `stat` quella sulla nostra macchina.

```
$server_fsize = $ftp->size($dlfile);
$ftp->get($dlfile,$dlfile);

use File::stat;
$st = stat($dlfile) or $newerr=1;
if ($newerr == 1){
    print "\nDownload Errato";
    print "\nIl file $dlfile non esiste!";
}else
    $local_fsize = $st->size;
    if ($server_fsize != $local_fsize) {
        print "\nDownload Errato";
        print "\nDimensione Discordante!";
    }else{
        print "\nDownload OK!";
    }
}
```

Quando vogliamo copiare sul Server FTP un file, usiamo la funzione `put()` in sovrascrittura sui file che hanno lo stesso nome, se però volessimo operare in aggiunta ad un file già esistente potremmo usare `append()`

Usando `binary()` trasferiamo file cosiddetti "binari" che non possono essere letti con dei comuni editor di testo, per tutti gli altri abbiamo `ascii()`

Quando ci loggiamo ad un Server FTP oltre al nome dell'Host possiamo definire anche delle opzioni:

Timeout	120 secondi di Default
Firewall	Nome di un Firewall FTP
Port	21 di Default
Debug	0 di Default, se uguale a 1 da le stesse Display di un FTP a linea di comando. Utile per capire se i comandi impartiti lavorano correttamente.
Passive	True o False, specifica se devono essere eseguiti trasferimenti in modalità passiva.

```
$hostFTP="ftp.miosito.it";
$ftp = Net::FTP->new($hostFTP,Debug=>1,Timeout=>20);
```

Infine oltre ai comandi `dir()`, `delete()`, `mkdir()`, `rmdir()` e `rename()` che hanno lo stesso utilizzo di quelli omologhi sul sistema abbiamo `quot()` per inviare dei veri e propri Comandi FTP definiti nella RFC 959.

```
# Posso cambiare gli attributi di un file via ftp
$ftp->quot('site','chmod 0777 miofile');
```

Nell'esempio proposto ci devono essere due requisiti fondamentali; il primo che il Server FTP sia Unix ed il secondo che l'Account (username/password) con cui entro non sia Anonymous.

### 19.6 World Wide Web

L'utilizzo del modulo LWP (Library WWW for Perl) ha un utilizzo ben più esteso di questo ma voglio introdurla perché ha una funzionalità molto utile che ci servirà in un esempio proposto nell'ultimo Capitolo.

Questa semplice funzionalità ci permette di scaricare da Internet una qualsiasi pagina HTML e posizionarla in un array.

```
use LWP::UserAgent;

$url = "http://www.google.it";
$browser = LWP::UserAgent->new();
$response = $browser->get($url);
@html = split("\n",$response->content);

foreach $rigaHTML (@html) {
    print "\n$rigaHTML";
}
```

`get()` e `content` sono i due Metodi del nuovo oggetto `$browser` istanziato, che ci permettono questa meraviglia.

## 19.7 Spedire una EMAIL

La gestione delle EMAIL utilizza due protocolli; il primo è l'SMTP (Simple Mail Transport Protocol) che serve per la spedizione ed il secondo è il POP3 (Post Office Protocol) che ne permette la ricezione.

```
use Net::SMTP;

$smtp = Net::SMTP->new($MailServer, Timeout=> 50, Debug=>0);
$smtp->mail($mittente);
$smtp->to($destinatario);
$smtp->data();
$smtp->datasend("To: $destinatario");
$smtp->datasend("From: $mittente");
$smtp->datasend("Subject: $oggetto");
$smtp->datasend("\n");
$smtp->datasend($testo);
$smtp->dataend();
$smtp->quit;
```

Il \$MailServer è il nome del Server che il vostro Provider Internet vi ha dato per poter spedire la posta, ad esempio Virgilio usa out.virgilio.it

data() che apre il canale per inviare dati col datasend() finché non arriva il dataend() che impacchetta tutto e lo spedisce al nostro MailServer. In questo modo abbiamo spedito una EMAIL di puro testo (quella che i Client di posta chiamano Plain Text). Ma cosa succede se vogliamo pure allegare un file?

Se è un file Ascii non c'è problema perché il datasend() lo spedisce tranquillamente (magari aggiungendo qualche backslash per le virgolette). Se però abbiamo dei file binari come gli eseguibili e i compressi dobbiamo servirci di un'estensione dello Standard MIME (Multi-purpose Internet Mail Extension) denominata Base64 che li converte in file Ascii.

```
use MIME::Base64;
use MIME::QuotedPrint;
use File::stat;

$FileIN = $ARGV[0];
$st = stat($FileIN) or die "\nFile non trovato";
$size = $st->size;
open(FILEIN, "+<$FileIN");
open(MIME, ">$FileMIME");
binmode(FILEIN); # Solo su Windows altrimenti va tolto
$num = read(FILEIN, $buffer, $size);
$encoded = encode_base64($buffer);
print MIME $encoded;
close MIME;
```



```
close FILEIN;
```

`encode_base64()` trasforma un file Binario in un file Ascii pronto per essere scritto su disco per poi essere recuperato e spedito col `datasend()`

Se del nostro messaggio originale non vogliamo "perdere" dei caratteri speciali come le accentate allora useremo la funzione `encode()` del `MIME::QuotedPrint`.

```
use MIME::QuotedPrint();
open(MIME, "<<$FileMIME");
while (!eof(MIME)){
    $row= MIME::QuotedPrint::encode(<MIME>);
    $smtp->datasend($row) ;
}
```

Mi auguro con tutto il cuore che questi esempi vengano usati per applicazioni serie e non per aumentare il già rilevante spamming di MAIL su Internet.

## 19.8 Ricevere una EMAIL

La ricezione di una MAIL incorpora una gestione vera e propria. Una volta connessi al proprio POP Server ed inserito l'Account personale (Username e Password)

```
use Net::POP3;
$pop3 = Net::POP3->new($POPServer);
$totMsg = $pop3->login($username,$password);
```

abbiamo nello scalare `$totMsg` l'elenco delle MAIL in attesa di essere scaricate.

Possiamo decidere di prendere una MAIL per volta con la funzione `get()`, salvarcela su di un file e cancellarla dal Server con `delete()`

```
Open(EMAILS, ">>email.txt");
for($i=1;$i<=$totMsg;$i++) {
    $email_ref = $pop3->get($i);
    print EMAILS @$email_ref;
    $pop3->delete($i);
}
close EMAILS;
```

Oppure possiamo semplicemente visualizzarne l'anteprima solo per vedere se ci sono delle novità. La funzione `top()` vuole il numero del messaggio ed il numero delle righe del corpo del messaggio che ci interessa vedere.

```
for($i=1;$i<=$totMsg;$i++) {  
    $email_ref = $pop3->top($i,0);  
    ($subject, $from, $data) = MailHeader($email_ref);  
    print "\n$data,$from,$subject";  
}
```

sia `get()` che `top()` resituiscono il Reference di un Array per cui fate attenzione alla differenza tra `$email_ref` e `@$email_ref`; il primo è il Reference ed il secondo è l'Array vero e proprio.

Terminando la sessione con `quit()` cancelliamo fisicamente dal Server le MAIL che abbiamo contrassegnato con il `delete()`

```
$pop3->quit();
```

## 19.9 In Conclusione

---

Con questo capitolo si conclude il corso vero e proprio, ne restano quattro per gli esempi pratici in cui applicheremo quello che abbiamo appreso fino a questo momento.

Buon divertimento!

# 20

## Esempi Pratici Parte I: Network Scanner

### 20.1 La Scansione della LAN

---

Quando si deve gestire una LAN molto spesso capita di doverla monitorare per vedere se tutti gli Host sono raggiungibili, se certi servizi sono attivi etc, etc... La faccenda però si complica quando non abbiamo un unico segmento (router) ma diversi dislocati altrove, trasformando una LAN (Local Area Network) in una WAN (Wide Area Network) ovvero quando una Rete Locale diventa Geografica. Per fare questo esistono tantissimi prodotti in commercio ma anche altri OpenSource, tra questi il più noto è NMAP (assurto alla cronaca per essere stato il tool usato da Trinity in Matrix Reloaded) NMAP esiste anche in versione Win32 ma se avete installato il Service Pack #2 non vi funziona più; a detta di Microsoft NMAP è un tool che non è poi così utile alle persone rispettabili, anzi chi lo usa lo fa perché è sicuramente un Cracker (Hacker maligno).

### 20.2 L'Idea di Base

---

Togliamoci completamente dalla testa l'idea di un tool alla NMAP – anche se esistono moduli aggiuntivi per Perl – l'unica cosa che possiamo costruirci è una piccola utility che ci permetta, all'occorrenza, di scansionare da un singolo indirizzo IP ad un'intera SubNet, fino al controllo quotidiano dei soli Host elencati in un file Ascii.

Una SubNet è un intervallo di indirizzi che fanno parte della stessa Classe. Per Classe intendo tutti gli indirizzi IP che possono essere contenuti in una Rete con mascheratura (SubNet Mask) 255.255.255.0, ovvero 255 da 1 a 254. Questa che viene definita Classe C (vi assicuro che non è una Mercedes) ed è la classica LAN aziendale e casalinga.

### 20.3 The Well-Know TCP Ports

---

Le porte TCP “ben conosciute” sono le prime 1024 (e sono state assegnate dall’Internet Assigned Numbers Authority) anche se in realtà quelle realmente “famoso” sono poco più di un centinaio e sono quelle contenute nel file `services` presente in ogni Sistema Operativo. Quelle di queste ancora più note le ho messe all’interno di un Hash

```
%Porte = (
    13 => 'daytime',
    21 => 'ftp',
    22 => 'ssh',
    23 => 'telnet',
    25 => 'smtp',
    37 => 'time',
    53 => 'DNS',
    80 => 'http',
    110=> 'pop3',
    137=> 'netbios',
    138=> 'netbios',
    139=> 'netbios-ssn',
    445=> 'microsoft-ds');
```

In realtà ogni macchina ha a disposizione ben 65536 Porte per i propri servizi TCP/IP e se le volessimo scansionare tutte dovremmo mandare in loop non un semplice `%Port` ma un contatore. Vista la natura didattica dell’esempio ho optato per scansionare solo le “Well-Know” più famose contenute nel File `services` della macchina che lancia il `netscan.pl`

La funzione `GetFileServices()` non fa altro che leggere il file (che cambia posizione a seconda del Sistema Operativo) e riempire l’Hash `%Porte` sostituendosi alla precedente inizializzazione. Come vedete, un chiaro esempio di cross-platform.

### 20.4 I Moduli

---

I moduli che ci interessano sono:

- `Net::Ping`
- `IO::Socket`
- `Getopt::Std`

Il `Net::Ping` come dice il nome ci serve per verificare se una macchina della LAN è raggiungibile o meno, `IO::Socket` per effettuare la scansione delle porte, mentre `Getopt::Std` serve alla gestione dei parametri a riga di comando.

Il `Net::Ping` lo abbiamo ampiamente spiegato nel Capitolo sul Networking, non abbiamo invece visto `IO::Socket`, un modulo piuttosto complesso che

prevede una certa esperienza nella programmazione del Socket TCP/IP (un Socket è una sorta di mezzo per collegare tra loro un Client e un Server). In questo esempio però ne facciamo un utilizzo piuttosto semplice, ovvero proviamo ad aprire una certa porta di un indirizzo IP per vedere se è attiva.

```
$sock = IO::Socket::INET->new("$IpAddr:$Port");
if ($sock) {
    print "\nLa Porta $Port sull'IO $IpAddr è Aperta";
}else {
    print "\nLa Porta $Port sull'IO $IpAddr è Chiusa";
}
```

Ed è quello che fa la funzione PortScan()

Getopt::Std è molto utile nella gestione degli switch a riga di comando che non siano posizionali come @ARGV. Se lancio uno script in questa maniera:

```
# perl netscan.pl host 10.14.1.1
```

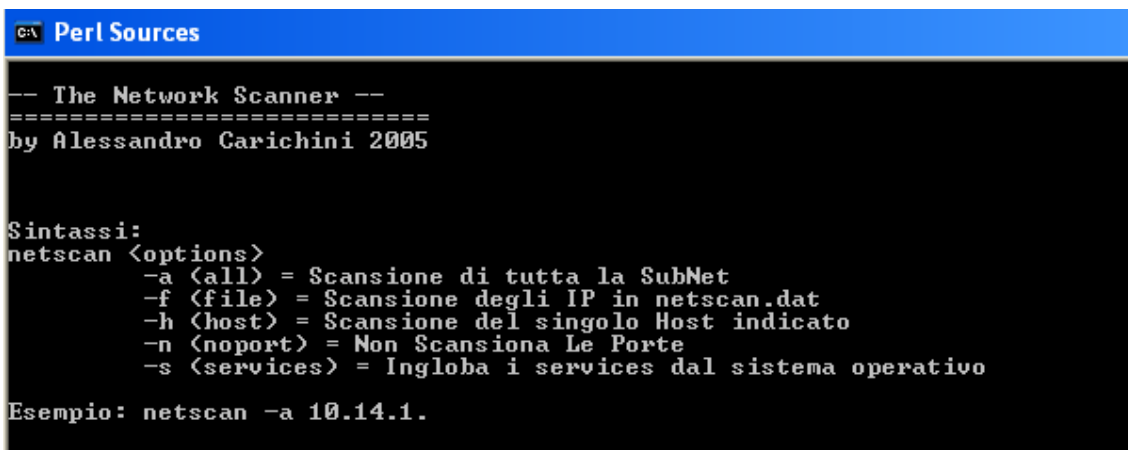
Avrò \$ARGV[0] = "host" e \$ARGV[1] = "10.14.1.1" se però comincio a voler inserire swicth opzionali con parametri a seguito allora dovrò definirli in questa maniera:

```
getopts("a:fh:ns", \%options);
```

Sostanzialmente riempio l'Hash %options con gli switch "-a", "-f", "-h", "-n" e "-s" non appena ne incontro uno. Per quelli con i due punti a seguito registro anche l'opzione supplementare.

```
if ($options{a}) {
    print "\nHo trovato lo switch -a contenente " . $options{a};
}
```

## 20.5 Il Programma



```

-- The Network Scanner --
=====
by Alessandro Carichini 2005

Sintassi:
netscan <options>
  -a <all> = Scansione di tutta la SubNet
  -f <file> = Scansione degli IP in netscan.dat
  -h <host> = Scansione del singolo Host indicato
  -n <noport> = Non Scansiona Le Porte
  -s <services> = Ingloba i services dal sistema operativo

Esempio: netscan -a 10.14.1.
```

Il fulcro del programma sta tutto in questa routine:

```
if (HostExist($ip)){
    print "OK ! Host $ip raggiungibile!";
    foreach $Port (keys(%Porte)) {
        $PortName = $Porte{$Port};
        if (PortScan($IpScan,$Port)) {
            print "\n\t $Port:$PortName";
        }
    }
}else {
    print "Host NON RAGGIUNGIBILE";
}
```

HostExist() verifica che l'Indirizzo IP sia raggiungibile, in quel caso effettua un PortScan() di tutte le porte listate in %Porte, se le trova attive (in ascolto o listening) lo segnala, altrimenti prosegue.

Se volete scansionare la macchina Windows da cui lanciate lo script, noterete la presenza delle Porte 25 e 110 in Listening. Questo dipende dalla presenza di un Antivirus che per controllare la posta in entrata (POP3->110) e quella in uscita (SMTP->25) le deve mantenere costantemente aperte, seppur dietro al firewall.

## 20.6 In Cosa lo Possiamo Migliorare?

- Verificare la correttezza dell'Indirizzo Host e l'Indirizzo di SubNet; potrei infatti scrivere un indirizzo formalmente errato (10,14.1.0 anziché 10.14.1.1) per un Host o per una SubNet (10.14.1 anziché 10.14.1.)
- Se si scansiona la SubNet ad ogni lasso di tempo si possono verificare gli Host che non sono più raggiungibili
- Aggiungere un nuovo switch per scansionare tutte le SubNet contenute in un file Ascii (come per i singoli Host).

Se poi avete altre esigenze fatevi avanti....

# 21

## Esempi Pratici Parte II: Il Formato CSV

### 21.1 Il Common Separated Value

---

Il CSV - o come lo chiama Excel, "Delimitato dal Separatore di Elenco" - è un formato ormai divenuto universale per il trasporto dei dati in formato tabella. Per la verità nessuno lo ha mai descritto formalmente, ma da quando è comparso dalla prima versione di VisiCalc (il papà di Excel, appunto) tutti lo hanno subito adottato.

Il motivo del successo è nella semplicità del nome stesso, ovvero dei Valori Separati da una Virgola (VSV per i non anglofoni). A dirla tutta nel corso degli anni ha subito qualche variante, difatti alcuni (come il sottoscritto) preferiscono sostituire la virgola col punto e virgola poichè nella notazione italiana non viene usata come separatore dei decimali. Anche per questo preferisco l'acronimo Common Separated Value visto che dà più l'idea di un formato flessibile ad ogni esigenza.

Ma di che cosa stò parlando? Beh, scusate il preambolo per descrivere il semplice modo di rappresentare un record per riga ed i suoi campi per colonna separati da una virgola, ovvero:

```
NR,NOMINATIVO,PROFESSIONE,AZIENDA,CODICE
1, "Alessandro Carichini", "Programmatore", "Manuali.Net", 9282828
```

Come potete vedere dall'intestazione (la prima riga) sono indicati i nomi dei campi corrispondenti ad ogni riga successiva.

### 21.2 L'Installazione del Modulo

---

L'unico modulo che serve installare è il `Text::CSV_XS` presente sia nel repository CPAN che ActiveState

- Da Linux (CPAN)  

```
# perl -MCPAN -e "install Text::CSV_XS"
```
- Da Win32 (PPM)

```
C:\>ppm install Text-CSV_XS
```

### 21.3 La Lettura di un file CSV

Un File CSV si apre come un normalissimo File di Testo

```
open (FILECSV, "<$FileCSV") or die "\nFile non trovato!\n";
```

è attraverso il modulo `Text::CSV_XS` che avviene il parsing vero e proprio. Per prima cosa creiamo l'oggetto `$csv` col Metodo `new` specificando alcune costanti qualora non fossero i valori di default (come invece quelli presentati)

```
$csv = Text::CSV_XS->new({
    'quote_char' => '"',
    'escape_char' => '"',
    'sep_char' => ',',
    'binary' => 0});
```

<code>quote_char</code>	il Carattere per rappresentare i campi a blank
<code>escape_char</code>	Il Carattere di escape; se all'interno di un testo mi trovo dei doppi apici, dovrò scrivere "" per evitare che interferiscano con quelle della stringa vera e propria
<code>sep_char</code>	Il Carattere Separatore dei campi; trattandosi di CSV è quindi una virgola
<code>binary</code>	Se è 1 posso includere anche caratteri binary all'interno di un campo

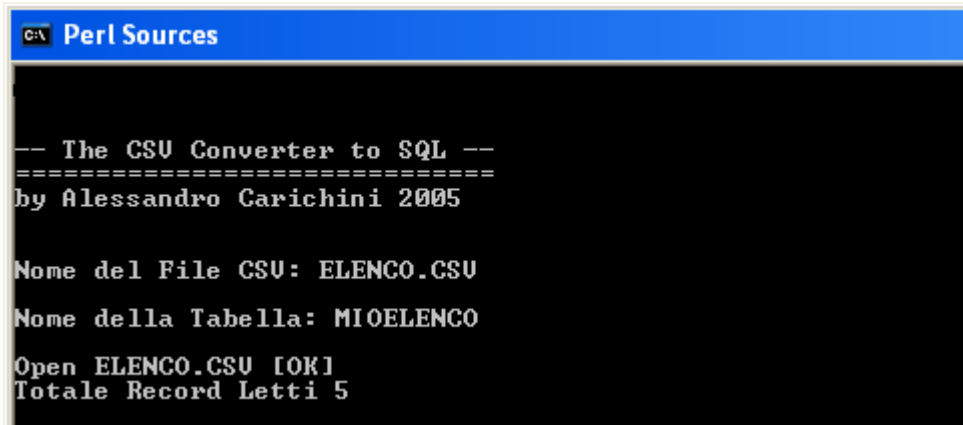
```
while (!eof(FILECSV)){
    $buff = <FILECSV>;
    if ($csv->parse($buff)) {
        @fields_parsed = $csv->fields();
        foreach $field (@field_parsed){
            print $field . ",";
        }
        print "\n";
    }
}
```

`parse()` verifica che si tratti di un record CSV mentre `field()` riempie un array per ogni virgola trovata (o `sep_char` indicato).



## 21.4 Il Programma

Si tratta di un utility che prende in input un file CSV e lo trasforma in un file di testo contenente i comandi SQL per l'inserimento dei records. Molto utile da integrare in operazioni di Import/Export da un RDBMS ad un altro.



```

c:\ Perl Sources
-- The CSV Converter to SQL --
=====
by Alessandro Carichini 2005

Nome del File CSV: ELENCO.CSV
Nome della Tabella: MIOELENCO

Open ELENCO.CSV [OK]
Totale Record Letti 5
    
```

Il programma non è auto-configurante e quindi va adattato a seconda della situazione poiché da per scontato alcune cose:

- La prima riga del file CSV deve fungere da descrittore dei campi per cui se il vostro estrattore CSV di fiducia la omette, ve la dovete scrivere a mano
- Il Comando SQL per l'inserimento di un record è `INSERT INTO (CAMPI) VALUES (CONTENUTO)` che è sì standard per l'80% degli SQL ma magari non per il 20% che interessa a voi

E' interessante notare che uso una RegExpr per verificare se un campo è alfanumerico o meno per l'aggiunta del singolo apice (che indica le stringhe in SQL)

```

foreach $field (@fields) {
    if ($field =~ /[\\w]/){
        $field = "'$field'";
    }
    $$SQLValue = $$SQLValue . "$field,";
}
    
```

## 21.5 In Cosa Lo Possiamo Migliorare?

- Aniché scrivere il comando SQL per esteso da dare in pasto all'RDBMS sarebbe interessante utilizzare il DBI per inviarlo direttamente al DB interessato; magari utilizzando i più noti: Oracle, MySQL, ec....
- Aggiungendo più informazioni nella riga di intestazione del CSV si potrebbe ricavare pure il comando SQL per la creazione della Tabella vera e propria.

- Quando fa l'INSERT INTO verifica solo che un campo sia numerico o alfanumerico; alcuni SQL utilizzano funzioni particolari per le date ed i BLOB

Cos'altro dire? Che il programma è in continua evoluzione....

# 22

## Esempi Pratici Parte III: Quote of the Day

### 22.1 La Citazione del Giorno

---

E' uno degli script più gettonati poichè capita spesso di fare un sito (siamo tutti WebMaster) e di volerci mettere in qualche angolino la citazione del Giorno presa dal proprio archivio (ebbene sì lo ammetto, ma il mio è fatto in PHP). Perché allora non costruirci un'applicazione ?

### 22.2 La Lista della Spesa

---

I Moduli che ci interessano sono sostanzialmente quelli per il database che sarà MySQL anche perché è il più presente nei contratti di hosting dei siti Internet. Per cui useremo:

- DBI
- DBD::mysql

E visto che vogliamo fare le cose per bene, formattiamo l'output in un certo modo con:

- Text::Wrap

### 22.3 Creazione del Database

---

Da linea di comando (sia per Linux che per Windows):

```
# mysql -u root -p < dbquotes.sql
```

Vi chiederà la password di root (o invio se non l'avete impostata) e creerà tutta la struttura del DB per poter eseguire lo script `quotes_day.pl`

Nel file `dbquotes.sql` oltre ai comandi per la creazione del DB e dell'unica tabella (QUOTES) sono state inserite anche una quarantina di simpatiche citazioni prese dal mio "archivio personale" (come se le avessi scritte io).

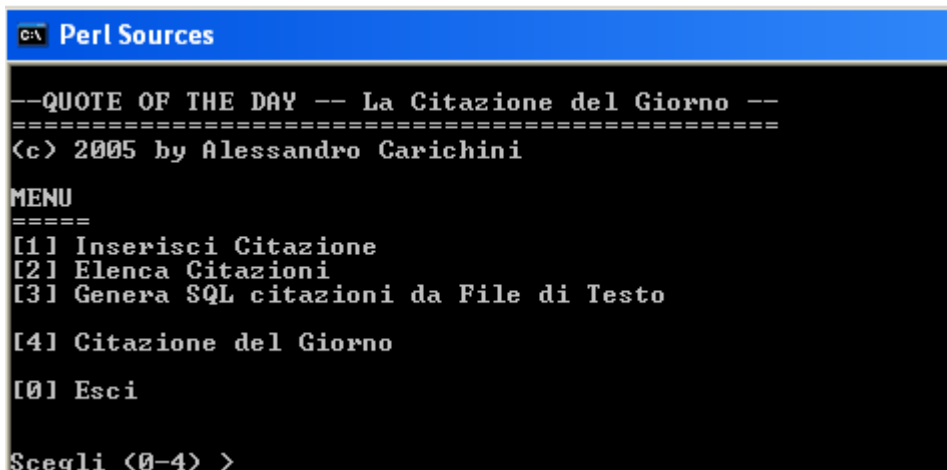
**Tabella QUOTES:**

Campo	Descrizione
ID_QUOTES	Identificativo della Citazione è auto_increment
QUOTES	Testo della Citazione
AUTHOR	Autore della Citazione
NOTES	Note aggiuntive riguardanti la citazione, ad esempio la sorgente da cui è stata tratta ec..

L'ID\_QUOTES è utile per avere l'Identificativo Univoco di un record senza dover fare una Query troppo complessa. Così che per cancellare una citazione basterà il comando SQL

```
DELETE FROM QUOTES WHERE ID_QUOTES = $numero
```

**22.4 Il Programma**



Dal Menu possiamo gestire il nostro DB delle Citazioni solo se aggiungiamo alla riga di comando lo switch menu (appunto) perché altrimenti visualizza direttamente la Citazione del Giorno (che è la funzione principale di questo script)

Per avere un valore casuale non ho utilizzato la rand() del Perl ma quella di MySQL che però non so quanto sia standard SQL

```
SELECT * FROM QUOTES ORDER BY RAND() LIMIT 1
```

ImportaTEXT() è una sub che legge in input un file Ascii contenente una Citazione per riga non nel formato CSV ma in quello definito a Lunghezza Fissa, ovvero ogni campo parte da una colonna prefissata. Nel nostro caso:

```
AUTORE (da posizione 1 a posizione 82)
TESTO (da posizione 83 in avanti)
```

Questo formato è semplice da costruire e da aggiornare con dei “copia e incolla” fatti da altri siti o altre fonti elettroniche - oops! ma forse questo infrange qualche copyright, ma io non ve l’ho detto.

### 22.5 In Cosa Lo Possiamo Migliorare?

---

- La possibilità di usarlo via Web e quindi interfacciarlo con il CGI; magari la parte di manutenzione farla rimanere a riga di comando mentre la Citazione del Giorno vera e propria in HTML
- Far esportare quotidianamente la “Citazione del Giorno” insieme al nome dell’Autore in un file di testo ed utilizzarlo come Firma (o Signature come indicano alcuni) nel proprio Client di EMAIL, avrete così ogni giorno una citazione diversa da allegare alla posta che spedite.
- Esportazione in formato CSV del DB
- Elenco ordinato per Autore
- Possibilità di cancellare una Citazione inserita erroneamente o di modificarne una esistente

E chi più ne ha più ne metta....

# 23

## Esempi Pratici Parte IV: Listino Cambi EURO

### 23.1 Da Un Semplice File XML...

---

La Banca Centrale Europea pubblica quotidianamente all'indirizzo:

<http://www.ecb.int/stats/exchange/eurofxref/html/index.en.html>

la lista delle Monete Estere più importanti con affiancato il relativo cambio in EURO. La pagina HTML oltre ad avere notizie interessanti è anche piena zeppa di altre informazioni poco utili (immagini, link, ecc..) per questo esiste un altro indirizzo che contiene le stesse informazioni in formato XML.

<http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml>

Il Formato XML è una sorta di CSV avanzato, infatti ha una struttura simile all'HTML (per l'utilizzo di Tag che ne definiscono la struttura) ma serve a contenere i record di un database. Sebbene il CSV sia un formato largamente riconosciuto ed utilizzato, l'XML ne sta diventando il degno sostituto.

### 23.2 L'Idea di Base

---

L'idea di base è quella di scaricarci quotidianamente il file XML ed importarlo in un database MySql così che lo si possa interrogare non solo per conoscere le quotazioni dell'Euro di fine giornata ma soprattutto per mantenere un archivio storico su cui fare statistiche e medie in un arco di tempo.

Per quanto riguarda la gestione del DB abbiamo i soliti moduli:

- DBI
- DBD::mysql

Per scaricarci il file XML dal Web invece abbiamo il modulo standard LWP (Library for WWW in Perl):

- LWP::UserAgent

### 23.3 Il Parsing dell'XML

Esistono parecchi Moduli Perl che gestiscono file XML anche se questo, in particolare, non può proprio definirsi un XML classico. Appartiene allo standard GEMES un formato per dati statistici sviluppato dall'Unione Europea che usa una struttura in stile XML (eXtensible Markup Language)

```
<Cube>
  <Cube time="2005-07-12">
    <Cube currency="USD" rate="1.2166" />
    <Cube currency="JPY" rate="135.33" />
    <Cube currency="CYP" rate="0.5736" />
    <Cube currency="CZK" rate="30.211" />
  </Cube>
```

Analizzando il file scaricato possiamo ricavarne alcune indicazioni, una su tutte il fatto che solo i tag che iniziano con <Cube> sono quelli che ci interessano. Questo riduce notevolmente la complessità del parser che ho creato all'interno della funzione ParseXML() che ci deve restituire solo tre campi:

- Data Estrazione (time)
- Divisa (currency)
- Cambio (rate)

Il \$value a zero (false) mi indica se nella riga XML datagli in pasto c'è qualcosa di buono. Grossomodo il funzionamento è questo:

```
$divisa,$value,$data = ParseXML($buffer)
if ($value){
    Print "OK la riga contiene dei Tag interessanti";
    if ($data){
        $dataXML = $data;
    }elseif($divisa) {
        # Aggiorna DB
    }
}else {
    print "Nulla di buono!";
}
```

Se uno qualsiasi degli scalari (\$divisa,\$value,\$data) mi ritorna con un valore diverso da zero significa che contiene un campo valido del flusso XML e quindi lo posso utilizzare. Una piccola parentesi sul campo \$data, come vedete ho dovuto fare in modo che la prima volta che viene rilevato lo si deve appoggiare su di un altro scalare per evitare di perdere il valore.

### 23.4 Creazione del Database

Il file euroexchange.sql è un file di testo contenente i comandi SQL per la creazione del Database e delle due Tabelle:

- CURRENCY
- EXCHANGE

CURRENCY contiene l'elenco delle monete con codice di tre caratteri e descrizione mentre EXCHANGE è il cambio giornaliero vero e proprio.

Facendo un semplice JOIN tra le due tabelle le possiamo mettere in relazione in base all'Identificativo Univico (ID\_CUR) della Divisa

```
SELECT A.ID_CUR,B.DES_CUR,A.RATE
FROM exchange A
JOIN currency B
ON B.ID_CUR = A.ID_CUR
```

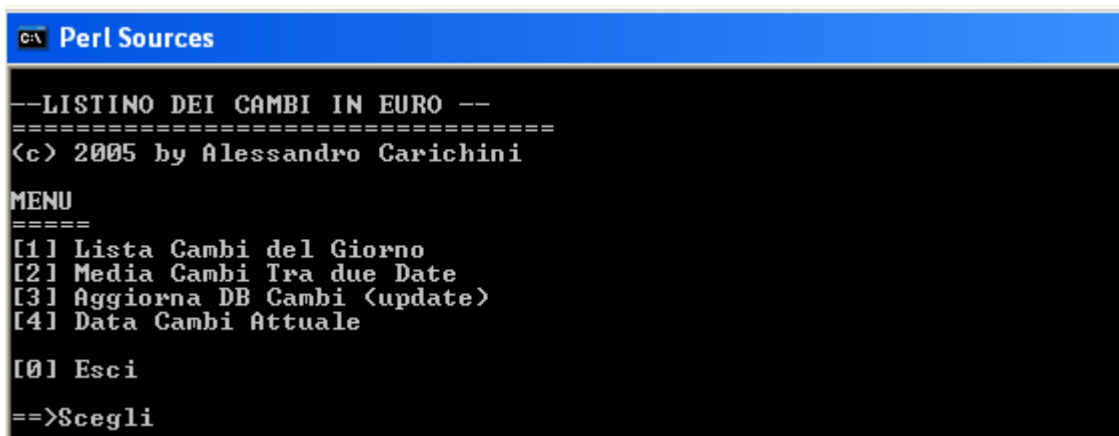
Quando in futuro, la Banca Centrale Europea aggiungerà nuove monete nella lista dei Cambi, vi troverete i soli 3 caratteri identificativi della divisa senza la corrispondente descrizione, dovete allora porre mano al MySQL per fare un INSERT INTO nella tabella CURRENCY.

Per creare tutto il DB invece basta lanciare da riga di comando (in Win32 accertatevi di avere il comando mysql nel PATH)

```
# mysql -u root -p < euroexchange.sql
```

Vi chiederà la password di root (o invio se non l'avete impostata) e creerà tutta la struttura del DB per poter eseguire lo script euroexchange.pl

### 23.5 Il Programma



```
Perl Sources
--LISTINO DEI CAMBI IN EURO --
=====
(c) 2005 by Alessandro Carichini

MENU
=====
[1] Lista Cambi del Giorno
[2] Media Cambi Tra due Date
[3] Aggiorna DB Cambi (update)
[4] Data Cambi Attuale

[0] Esci

==>Scegli
```

Se a linea di comando aggiungete lo switch update bypassiamo il Menu Principale ed andiamo direttamente alla scelta n.3 ed usciamo dal programma.



Per il resto le options sono piuttosto esplicative, non vi resta che provarlo ed apporvi qualche modifica.

### 23.6 In Cosa Lo Possiamo Migliorare?

---

- Sicuramente la possibilità di usarlo via Web e quindi interfacciarlo con il CGI; magari la parte di manutenzione farla rimanere a riga di comando (soprattutto per l'Aggiornamento) mentre quella di visualizzazione può essere portata tranquillamente in HTML.
- Fare in modo che l'aggiornamento avvenga automaticamente, utilizzando lo switch a riga di comando `update` per farlo lavorare in background ed utilizzando il `crontab` da Unix o l'`at` da Windows per lanciarlo ad orari prestabiliti.
- Permettere l'inserimento di nuove Divise nella tabella CURRENCY senza doverlo fare manualmente su MySQL
- Visualizzare solo quelle Divise che interessano veramente
- Lo Scrolling video non è dei migliori, infatti se avete un numero di righe inferiori a 35 non vedete i primi elementi. Sarebbe opportuno fare videate di 20/25 righe con pausa.

Gli spunti per migliorare il programma ci sono, lascio a voi l'onere.

# Test finale

1. Che differenza c'è tra un linguaggio interpretato ed uno compilato?
2. Quali sono gli elementi necessari per far girare uno script in Perl?
3. Su quali piattaforme può essere eseguito?
4. Differenza tra Scalari, Array ed Hash
5. Cosa s'intende per "Interpolazione di Stringa" ?
6. Qual è l'elemento di identificazione per la gestione dei file ?
7. Differenza tra Moduli Standard e Moduli CPAN
8. Quali sono gli elementi essenziali per interfacciare Perl ad un Database Relazionale come MySQL?

9. In che modo Perl e CGI si legano?

10. Creare uno script che, partendo dalla directory di default (in modo ricorsivo), raggruppi i files per estensione creando una semplice statistica utilizzando la dimensione ed il numero dei files. Esempio:

lista\_ext.pl

```

=====
ESTENSIONE      Nr.Files      %      Tot.Byte
=====
.SXW             1           1%      13350
.VPD             3           3%       834
.PL            34          38%     73157
.HTM             1           1%      1426
.DAT             3           3%      8584
.LOG            1           1%       142
.CSV            1           1%       256
.TMP            1           1%     63488
.SQL            4           4%      21065
.DOC            37          41%    7377058
.ZIP            2           2%     3303108
=====
TOTALE           89         100%   10862468
=====

```

# Risposte al test finale

1. Che differenza c'è tra un linguaggio interpretato ed uno compilato?

Questa domanda è un po' un colpo basso, in effetti nel primo capitolo non ne parlo esplicitamente ma lo lascio intuire. La differenza più semplice ed evidente è durante la fase di debug, in un linguaggio interpretato basta modificare il file ascii contenente lo script e rilanciare l'interprete, in un linguaggio compilato invece devo modificare il sorgente, ricompilarlo (trasformandolo in codice macchina) e se non ci sono errori formali (di sintassi) lo posso mandare in esecuzione. Quindi in un linguaggio interpretato lo script viene eseguito dall'interprete (nel nostro caso il perl), mentre in un linguaggio compilato il compilatore non esegue il programma che riceve in ingresso, ma lo traduce in linguaggio macchina.

2. Quali sono gli elementi necessari per far girare uno script in Perl?

Un sistema operativo funzionante e l'interprete perl. (Capitolo 1)

3. Su quali piattaforme può essere eseguito?

Su tutte le versioni di Microsoft Windows (dal 95 in avanti), su Linux (qualsiasi distribuzione), su Unix (AIX, Solaris, HP-UX, ecc..) e su Macintosh (Os e OsX). (Capitolo 1)

4. Differenza tra Scalari, Array ed Hash

Uno Scalare è una variabile semplice contenente un unico elemento (o numerico o alfanumerico), un Array è un insieme elementi (lista) indicizzati da un valore numerico, un Hash invece è sempre un insieme di elementi indicizzati da un valore alfanumerico (Array associativo). (Capitolo 4)

5. Cosa s'intende per "Interpolazione di Stringa" ?

Quel meccanismo che permette - attraverso l'uso dei doppi apici - di sostituire al nome della variabile il suo contenuto. (Capitolo 4)

6. Qual è l'elemento di identificazione per la gestione dei file ?

L'HANDLE che viene anche definito descrittore del file. (Capitolo 6)

7. Differenza tra Moduli Standard e Moduli CPAN

Da un punto di vista tecnico nessuno, infatti si tratta sempre di moduli esterni contenenti delle collezioni di funzioni, dal punto di vista funzionale invece i Moduli Standard sono inclusi nell'installazione del Perl mentre i moduli CPAN devono essere scaricati o dai vari mirror esistenti in rete o da repository di terze parti. (Capitolo 13)

8. Quali sono gli elementi essenziali per interfacciare Perl ad un Database Relazionale come MySQL?

Come indicato nel Capitolo 16, gli elementi essenziali sono i moduli DBI e DBD::mysql (scaricabili via CPAN) ed il client MySQL (attraverso il sito ufficiale di mysql)

9. In che modo Perl e CGI si legano?

Il CGI è un meccanismo (interfaccia) che permette ad un browser (il Client) di eseguire dei programmi remoti residenti su di un Server. Questi programmi possono essere scritti in Perl e vengono innescati dalle Form HTML del Client. (Capitolo 18)

10. Creare uno script che, partendo dalla directory di default (in modo ricorsivo), raggruppi i files per estensione creando una semplice statistica utilizzando la dimensione ed il numero dei files. Esempio: lista\_ext.pl

```
#!/usr/bin/perl
#-----
# Elenco Files Raggruppati per Estensione
# by Alessandro Carichini
# Corso Perl manuali.net
#-----

use File::Find;
use File::stat;

$start_dir='.';
$contafiles = 0;

find \&deep,$start_dir;

$sommaperc=0;
$sommasize=0;

printf("\n=====");
printf("\nESTENSIONE      Nr.Files      %      Tot.Byte");
printf("\n=====");

foreach $hext (keys(%hash_ext)){
    $TotExt= $hash_ext{$hext};
    $perc = sprintf "%2.3f", (($TotExt * 100) / $contafiles);
    $TotDim = $hash_dim{$hext};
    $sommaperc=$sommaperc+$perc;
    $sommasize=$sommasize+$TotDim;
    printf("\n%-10s %10d  %5d\%%  %10d", $hext, $TotExt, $perc, $TotDim);
}

printf("\n=====");
printf("\nTOTALE          %10d  %5d\%%  %10d", $contafiles, $sommaperc, $sommasize);

sub deep{
    ($fileName) = $_;
    if (-f) {
        $hst = stat($fileName) or die "No FILENAME $fileName";
        $fsize = $hst->size;

        $ext = uc(FindExt($fileName));
        $hash_ext{$ext}= $hash_ext{$ext}+1;
        $hash_dim{$ext}= $hash_dim{$ext}+$fsize;
        $contafiles++;
    }
}

sub FindExt{
    ($filename) = $_;
    $nPos = rindex($filename, '.');
```

```
return substr($filename,$nPos,length($filename)-npos);  
}
```

Chiaramente questo è solo uno spunto, il programma è migliorabile tanto che c'è pure un piccolo bug che vi sfido a trovare.