Formation Perl

Version Beta 0.0.1

Support Instructeur

Eric BERTHOMIER

9 mai 2005

Table des matières

Chapitre 1

Installation de Perl avec MySql à partir de Knoppix

1.1 A propos

1.1.1 Mots clés

perl Practical Extraction and Report Language

1.2 Installation de KNOPPIX 3.2 sur le disque dur

`http://forums.knoppix-fr.org/viewtopic.php?t=2545`

1.3 Installation des packages supplémentaires

La commande d'installation de package sous Debian (Knoppix) est `apt-get install <nomdupackage>`. La commande de recherche d'un package est `apt-cache search <nom du package>`. Il est bien sûr possible de combiner cette recherche avec un `grep`.
Pour réaliser des accès avec des bases de données type mysql il est nécessaire d'installer les packages suivants :

1.3.1 Perl

- perl
- perl-base
- perl-doc
- perl-modules
- libperl5.8

1.3.2 mysql

- mysql-server
- mysql-admin

`mysql-admin` n'est pas obligatoire mais permet de contrôler de manière graphique l'installation et la modification des différences bases sur votre serveur.

1.3.3 DBI

- libdbi-perl
- libdbd-mysql-perl

1.3.4 GD

- libgd-perl
- libgd-graph-perl
- libgd-gd1-perl

Chapitre 2

Expressions Régulières

2.1 A propos

2.1.1 Mots clés

perl Practical Extraction and Report Language

2.2 Exercices

2.2.1 Adresse IP

A l'aide de la commande `ifconfig`, retrouvez les différentes adresses IP de votre machine.

Listing 2.1 – ip.pl

```
1 #!/usr/bin/perl -w
2 use strict;
3
4 my @tabIp = `ifconfig `;
5
6 foreach (@tabIp) {
7         print "Adresse IP : $1\n" if ( /inet adr:(.*?)\s/ );
8 }
```

2.2.2 Qui est où ?

A l'aide de la commande `who`, établissez un rapport indiquant le nom de l'user suivi des consoles sur lequel il est connecté.

Exemple :
`eric -> :0, pts/1, pts/2, pts/0, pts/3, pts/4`

Listing 2.2 – quiestou.pl

```
1 #!/usr/bin/perl -w
2 use strict;
3
4 my %qui;
```

```
5  my @tab=`who`;
6
7  foreach (@tab) {
8          if ( /^(\w*)\s*(.*?)\s/ ) {
9                  if ( exists $qui{$1} ) {
10                         $qui{$1} .= ", $2";
11                 } else {
12                         $qui{$1} = "$2";
13                 }
14         }
15 }
16
17 while ((my $cle, my $valeur) = each (%qui) ) {
18         print "$cle -> $valeur\n";
19 }
```

Chapitre 3

Utilisation des modules en Perl[2]

3.1 A propos

3.1.1 Mots clés

PERL	Practical Extraction and Report Language.
use	commande perl permettant d'insérer un module.
our	commande perl permettant d'indiquer une donnée partagée.
qw	quoted word
@EXPORT	Liste des données exportées par défaut.
@EXPORT_OK	Liste des données exportée (import manuel).
@INC	Liste des répertoires de recherche des modules (équivalent au PATH Unix).

Cette documentation s'inspire librement des écrits de Sylvain Lhuillier.

3.2 Fichier module

3.2.1 Caractéristiques

- Un module est un fichier dont le suffixe est ".pm".
- Par convention, le nom du fichier doit commencer par une Majuscule.
- Le fichier doit se trouver dans un des répertoires indiqués par @INC (par exemple le répertoire local `.`).
- La première ligne d'un fichier module doit indiquer le nom du module **qui doit être identique** au nom du fichier sans extension à l'aide de la commande `package Nomdumodule`.

3.2.2 Exemple de fichier module

Listing 3.1 – Outils.pm

```
1 package Outils;
2 use strict;
3
4 sub bonjour {
5         print "Bonjour␣tout␣le␣monde␣!\n"
6 }
```

```
7
8 1;
```

La dernière ligne du script (`1 ;`) indique que le module s'est correctement chargé (VRAI). Dans le cas où une valeur de retour de 0 est retournée, le programme qui requiert ce module s'interrompt avec une erreur.

3.3 Utilisation d'un module

Il est possible d'utiliser le module en l'incluant par la commande `use`.

Listing 3.2 – script1.pl

```
1 use strict;
2 use Outils;
3
4 Outils::bonjour;
```

La commande `Outils::bonjour` correspond à l'appel de la fonction bonjour du module Outils.

3.4 Variables

Les variables des modules sont déclarables de 2 façons :
– les variables internes au module que l'on dira privées
– les variables accessibles de l'extérieur du module que l'on dira publiques

3.4.1 Variables privées

Une variable privée dans un module est déclarée à l'aide de la commande `my` suivi du nom de la variable.

Listing 3.3 – Outils.pm

```
1  package Outils;
2  use strict;
3
4  my $who = ''Eric'';
5
6  sub bonjour {
7          print "Bonjour␣$who␣!\n"
8  }
9
10 1;
```

Cette variable n'est pas accessible à l'extérieur du module. Il est à souligner qu'il est possible d'utiliser la commande `my` sur des fonctions de manière à rendre celle-ci privées.

3.4.2 Variables publiques

Une variable publique dans un module est déclarée à l'aide de la commande `our` suivi du nom de la variable.

Listing 3.4 – Outils.pm

```
1 package Outils;
```

```
2  use strict;
3
4  our $who = ''Eric'';
5
6  sub bonjour {
7          print "Bonjour $who !\n"
8  }
9
10 1;
```

Cette variable est accessible à l'extérieur du module.

Listing 3.5 – script2.pl

```
1  #!/usr/bin/perl -w
2
3  use strict;
4  use Outils;
5
6  $Outils::who=''Souricier'';
7  print "Bonjour $Outils::who";
```

3.5 Modules avec un nom composé

Les modules avec un nom composé permettent de regrouper des modules sous une même bannière. Par exemple Net inclus les modules Cmd, Config, Domain, FTP, hostent, netent, Netrc, NNTP, Ping, POP3, prtoent, servent, SMTP, Time.

3.5.1 Exemple : Création du Module Boite : :Outils

1. `mkdir Boite`
2. mv Outils.pm Boite/

Il nous faut maintenant modifier le module pour indiquer son nom complet devenu `Boite::Outils`.

```
package Boite::Outils;
```

Listing 3.6 – script3.pl

```
1  #!/usr/bin/perl -w
2  use strict;
3  use Boite::Outils;
4
5  $Boite::Outils::who="Souricier";
6  print "Bonjour $Boite::Outils::who";
```

Attention : la liste @INC doit contenir le répertoire contenant le module en question. Si nécessaire, il faut donc ajouter la ligne suivante : `push @INC, ''repertoire''` avant l'appel du module.

3.6 Blocs BEGIN et END

Les blocs BEGIN et END (`awk`) sont exécutés respectivement au chargement du module et à la fin de l'exécution du module. Ces deux blocs sont maintenant rendus inutiles par la programmation Objet (constructeur et destructeur).

Listing 3.7 – Outils.pm

```
1  package Outils;
2  use strict;
3
4  our $who = "Eric";
5
6  sub bonjour {
7          print "Bonjour␣$who␣!␣\n";
8  }
9
10 BEGIN
11 {
12         print "Chargement␣du␣module\n";
13 }
14
15 END
16 {
17         print "Fin␣du␣module\n";
18 }
19
20 1;
```

3.7 Export

Le terme `export` signifie l'exportation dans l'espace de nom du programme appelant des variables / fonctions du module.
Ce principe permet de ne pas avoir à utiliser la syntaxe `Module::Variable` ou `Module::Fonction` à chaque utilisation d'un élément du module mais simplement Variable ou Fonction.

Pour pouvoir exporter des éléments, notre module doit comporter les éléments suivants :

```
package Outils;
use Exporter;
our @ISA=qw(Exporter);
```

Le ligne `our @ISA=qw(Exporter);` signifie que l'on va réaliser des exports de type ISA (voir programmation objet pour les autres types d'export).

3.7.1 Types de variables d'un module

On différencie 3 types de variables dans un module :

1. les variables exportées par défaut. Le script appelant n'a rien à faire pour utiliser les éléments ainsi exportés, ces éléments se trouvent dans l'espace de nom du script dès l'utilisation du module.
2. les variables ou groupes (*tags*) de variables exportés. Le script appelant doit notifier ce qu'il désire importer dans son espace de nom.

3. les variables non exportables. Elles ne sont alors pas utilisables par le script (sauf de manière explicite Module : :...).

Les 2 premiers types de variables sont déclarées à l'aide de la fonction `our`.

3.7.2 Export par défaut

Les symboles exportés par défaut doivent être listés dans le tableau `@EXPORT`.

```
our @EXPORT=qw(&bonjour, &fonction, $variable);
```

Le & permet de désigner une fonction, il n'est pas obligatoire mais demeure souvent pour des raisons de lisibilité évidentes.

3.7.3 Export individuel

Les symboles exportés de manière individuelle sont importés par le programme appelant de manière explicite et en fonction de ses besoins. La déclaration au niveau du module s'effectue à l'aide du tableau `@EXPORT_OK`

```
our @EXPORT_OK=qw(&okbonjour, &okfonction, $okvariable);
```

Du côté du programme appelant, celui-ci doit indiquer les variables/fonctions qu'il désire importer du module.

Listing 3.8 – Outils.pm

```
1 use Boite::Outils qw(:DEFAULT &okfonction &okvariable);
2 use strict;
3
4 ...
```

Il est nécessaire d'utiliser la variable `:DEFAULT` pour indiquer que l'on désire importer en plus des variables/fonctions par défaut, les éléments exportés par défaut (`@EXPORT`).

3.7.4 Export par groupe

Il est possible de grouper de variables dans un ensemble. Chaque ensemble sera défini comme un élémént d'une table de hashage nommé %EXPORT_TAGS.

```
our %EXPORT_TAGS=(LISTE1=>[qw(&bonjour, &fonction)],
  LISTE2=>[qw($variable, $var2)]);
```

Par convention, le nom des tags est fixé en majuscule.

Attention : Les symboles pointés par les différentes Tags DOIVENT être déclarés dans les variable @EXPORT et/ou @EXPORT_OK. Donc la fonction `bonjour`, la variable `$variable` doivent être déclarés dans la table @EXPORT et/ou @EXPORT_OK.

L'import dans le script se fait de la façon suivante :

```
use Boite::Outil qw(:DEFAULT :LISTE1);
```

3.8 Exemple complet

Attention Suivant l'exemple précédent, le fichier Outil.pm se situe dans le répertoire Boite.

Listing 3.9 – Outil.pm

```
1  package Boite::Outils;
2  use strict;
3  use Exporter;
4
5  our @ISA=qw(Exporter);
6  our @EXPORT=qw($who);
7  our @EXPORT_OK=qw($qui &bonjour &aurevoir);
8  our %EXPORT_TAGS=(
9          fonctions=>[qw(&bonjour &aurevoir)],
10         variables=>[qw($who $qui)]
11 );
12
13 our $who = "Eric";
14 our $qui = "Toto";
15 my $prive = "Prive";
16
17 sub bonjour {
18         print "Bonjour␣$who␣!␣\n";
19 }
20
21 sub aurevoir {
22         print "Au␣revoir␣$qui␣!\n"
23 }
24
25
26 sub prive {
27         print "$prive";
28 }
29
30 BEGIN
31 {
32         print "Ceci␣est␣ééexcut␣au␣chargement␣du␣module\n";
33 }
34
35 END
36 {
37         print "Ceci␣est␣ééexcut␣à␣la␣fin␣de␣l\'éexcution␣du␣module\n";
38 }
39
40 1;
```

Listing 3.10 – script.pl

```
1  #!/usr/bin/perl -w
2
3  use strict;
4  use Boite::Outils qw( :DEFAULT $qui :fonctions);
5
6  print $who."\n";
7  print $qui."\n";
```

```
8 bonjour;
```

Chapitre 4

Le CGI avec PERL

4.1 A propos

4.1.1 Mots clés

PERL Practical Extraction and Report Language.

4.2 Rappel sur le HTML

Un formulaire se compose d'une balise `<FORM>` et se termine par une balise </FORM>. Ces deux balises permettent de placer des champs de saisie de données qui seront envoyés lors de l'auppui sur un bouton généré par la balise de type `INPUT TYPE=SUBMIT`.

Ce bouton `INPUT TYPE=SUBMIT` peut être remplacé par une IMAGE, c'est ce que nous utiliserons dans l'exemple ci-dessous pour illustrer le comportement de method=POST et method=GET.
A noter que lors de l'utilisation de cette image les coordonnées du clic sur l'image sont envoyées en plus du reste des informations.

4.3 Utilisation simple du formulaire

4.3.1 Le fichier HTML [1]

Listing 4.1 – postget.html

```
1  <HTML>
2  <HEAD>
3  <TITLE> Perl Example HTML </TITLE>
4  </HEAD>
5  <BODY>
6  <H2>Form Posts to a Perl CGI script</H2>
7  <HR>
8  <FORM ACTION="/cgi-bin/position.pl" METHOD="POST">
9  POST Method : Click within the image to obtain the x,y coordinates for that point.
10 <P>
11 <INPUT NAME="image" TYPE="image" SRC="linux.gif" >
```

```
12 </FORM>
13 <BR>
14 <FORM ACTION="/cgi-bin/position.pl" METHOD="GET">
15 GET Method : Click within the image to obtain the x,y coordinates for that point.
16 <P>
17 <INPUT NAME="image" TYPE="image" SRC="linux.gif" >
18 </FORM>
19 <HR>
20 </BODY>
21 </HTML>
```

4.3.2 Le fichier PERL

Listing 4.2 – position.pl

```
1  #!/usr/bin/perl -w
2  #
3  # Example of post-query in perl
4  #
5
6  print "Content-type: text/html\n\n";
7  print "<HTML><BODY> <H1>Mouse Button Query</H1>\n";
8  print "You submitted the following name/value pairs:\n";
9
10 if ($ENV{'REQUEST_METHOD'} eq "POST") {
11   $form = <STDIN>;
12
13 } else {
14   $form = $ENV{'QUERY_STRING'};
15 }
16 @pairs = split(/&/,$form);
17 print "<UL>";
18 while (@pairs) {
19   $pair=shift @pairs;
20   $pair =~ s/\+/ /g;
21   print "<LI><CODE>$pair</CODE>\n ";
22 }
23 print "</UL>";
24 print "\n</BODY></HTML>";
```

4.4 Formulaire avec champs de saisie

Listing 4.3 – formulaire.html

```
1 <html>
2 <head>
3 <title>Hummm, c&#39;est bon...
4 </title>
5 </head>
6 <body>
7 <h1>Salut, gourmand !</h1>
8 <form method="post" action="/cgi-bin/reponse.pl">
```

```
9  Quel est votre parfum favori ? <input type="text" name="parfum" />
10 <input type="submit"/>
11 </form>
12 </body>
13 </html>
```

4.5 Utilisation du module CGI

Listing 4.4 – parfum.pl

```
1  #!/usr/bin/perl -w
2
3  use CGI qw(:standard);
4
5  print header(),
6          start_html( -title => "Hummm, c'est bon...",
7                  -lang => "fr-FR"
8                  ),
9          h1("Salut, gourmand !");
10
11 if ( my $favorite = param ("parfum") ) {
12         print p("Votre parfum favori est $favorite.");
13 } else {
14         print start_form,
15         "Quel est votre parfum favori ? ", textfield('parfum'),"  ",
16         submit,
17         end_form;
18 }
19
20 print end_html();
```

4.6 Utilisation du module CGI version objet

Listing 4.5 – parfum_obj.pl

```
1  #!/usr/bin/perl -w
2
3  use CGI qw(:standard);
4
5  $q = new CGI;
6  print $q->header(),
7          $q->start_html( -title => "Hummm, c'est bon...",
8                  -lang => "fr-FR"
9                  ),
10         $q->h1("Salut, gourmand !");
11
12 if ( my $favorite = $q->param ("parfum") ) {
13         print $q->p("Votre parfum favori est $favorite.");
14 } else {
15         print $q->start_form,
16         "Quel est votre parfum favori ? ", $q->textfield('parfum'),"  ",
```

```
17              $q->submit,
18              $q->end_form;
19  }
20
21  print $q->end_html();
```

Chapitre 5

Utilisation de Perl avec MySql

5.1 A propos

5.1.1 Mots clés

perl	Practical Extraction and Report Language
DBI	Data Base Interface
GD	Perl Graphics Library

5.2 Création d'une base de données

Sous l'identité de root, tapez `mysql` puis

```
mysql> create database exemple;
Query OK, 1 row affected (0.00 sec)
mysql> use exemple;
Database changed
mysql> grant all privileges on exemple to eric;
Query OK, 0 rows affected (0.21 sec)
mysql> quit
Bye
```

5.3 Création / suppression d'une table

Listing 5.1 – Création / Suppression d'une Table

```
1  #!/usr/bin/perl -w
2
3  use DBI;
4
5  my $db = "DBI:mysql:exemple";
6  my $user = "eric";
7
8  my $dbh = DBI->connect($db,$user) ||
9          die "$0 : can't connect to $db : $DBI::errstr\n";
10
11 # Suppression de la table
```

```
12 my $requete = "DROP␣TABLE␣${user}_cd";
13 my $sth = $dbh->prepare ($requete) or die "$0:␣can't␣prepare:␣$DBI::errstr\n";
14 $sth->execute();
15
16 # éCration de la table eric_cd
17 $requete = "CREATE␣TABLE␣${user}_cd␣(
18 id␣INT,
19 title␣CHAR␣(50),
20 artist␣CHAR␣(50),
21 interprete␣CHAR␣(50),
22 maison␣CHAR␣(50),
23 year␣INT,
24 purchased␣DATE,
25 type␣CHAR␣(50),
26 nombre␣INT,
27 barcode␣CHAR␣(50),
28 prix␣FLOAT)";
29
30 $sth = $dbh->prepare ($requete) or die "$0:␣can't␣prepare:␣$DBI::errstr\n";
31 $sth->execute();
32
33 # Insertion de édonnes dans la table
34 $requete = "INSERT␣INTO␣${user}_cd␣VALUES␣(?,?,?,?,?,?,?,?,?,?,?)";
35 $sth = $dbh->prepare ($requete) or die "$0:␣can't␣prepare:␣$DBI::errstr\n";
36 $good=$bad=0;
37
38 open F, "cddata.txt";
39 while (<F>) {
40   chomp;
41   @cd = split /\|/;
42   $cd[6] = join '-', reverse split '/', $cd[6];
43
44   if ( $sth->execute(@cd[0..10])) {
45     $good++;
46   }
47   else {
48     warn "$0:␣can't␣insert␣$cd[0],␣$cd[1]:␣$DBI::errstr\n";
49     $bad++;
50   }
51 }
52 close (F);
53
54 # Fin d'utilisation de la base de édonnes
55 $dbh->disconnect;
56
57 # Rapport de fin
58 print "Added␣$good␣rows";
```

5.4 Lecture d'une base de données et affichage de statistiques

Listing 5.2 – Lecture d'une base de données et affichage de statistiques

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

```
2
3  use DBI;
4  use GD::Graph::pie;
5  use CGI;
6
7  # use diagnostics;
8  my $db = "DBI:mysql:exemple";
9  my $user = "eric";
10
11 my $dbh = DBI->connect($db, $user) ||
12         die "$0 : can't connect to $db : $DBI::errstr\n";
13
14 $requete = "select year(purchased) as y, count(*) from ${user}_cd group by y";
15 $sth = $dbh->prepare ($requete) or die "$0: can't prepare: $DBI::errstr\n";
16 $sth->execute() or die "$0: can't execute: $DBI::errstr\n";
17
18 my $cd_ref = $sth->fetchall_arrayref() or
19         die "$0:can't fetch data $DBI::errstr\n";
20
21 $sth->finish;
22 $dbh->disconnect;
23
24 # Separate labels and values
25 my $row="";
26 foreach $row (@$cd_ref) {
27         push @{$data[0]}, @$row[0];
28         push @{$data[1]}, @$row[1];
29 }
30
31 # Dessiner le graphisme
32 $mygraph=new GD::Graph::pie();
33 $mygraph->set(
34         title => 'CD Purchases by Year',
35         label => 'Copyleft Eric',
36         axislabelchr => 'black',
37         pie_height => 36
38 );
39
40 my $gdo = $mygraph->plot (\@data);
41
42 my $cgi = new CGI;
43 print $cgi->header(-type =>'image/png', -expires=>'+10');
44
45  Output img
46 print $gdo->png;
47
48 exit 0;
```

Listings

Bibliographie

[1] IBM. http ://www-306.ibm.com/software/webservers/dgw/apisamp.htm.

[2] Sylvain Lhuillier.

Index