La gestion des options en Perl

Options en Perl

On peut gérer deux types d'options :

- Les options composées d'un caractère, comme par exemple `-l` : utiliser le module `Getopt::Std` inclus dans Perl 5.x
- Les options avec nom long comme `--long` : utiliser le module Getopt::Long

  Non-standard mais bien plus pratique.

Getopt::Std

- Options composées d'un seul caractère,
- Introduites par un seul –
- Valeurs :
  - Booléennes (définies ou non – undef)
    exemple : -c
  - Quelconques : chaînes de caractères
    exemple : - a toto

Getopt::Std

Options définies à l'aide des fonctions:

- `getopt`

```
getopt("aBC");
getopt("aBC" , \%opts);
```

Chaque option attend une valeur associée. Si une option non définie est spécifiée, il n'y a pas de *warning* mais les options qui suivent sont ignorées

- `getopts`

```
getopts("aBC:") ;
getopts("aBC:", \%opts) ;
```

Les options suivies d'un `:` attendent une valeur associée, sinon ce sont des options booléennes.

Getopt::Std

- Pour les deux fonctions (`getopt` et `getopts`), si l'argument \%opts n'est pas spécifié, les variables `$opt_X` (où X est le nom de l'option) sont utilisées.
- Les variables `$opt_X` doivent être déclarées avec le mot-clé `our`.

Getopt::Std getopt exemple 1

Accès aux valeurs des options par les variables $opt_X

```
use Getopt::Std ;
our $opt_a;
our $opt_B;

getopt('aB') ;
if (defined $opt_a) {
      print "a: $opt_a\n";
} else { print "a undef\n";}
if (defined $opt_B) {
      print "B: $opt_B\n";
} else { print "B undef\n";}
```

Getopt::Std getopt exemple 2

Accès aux valeurs des options par %opts

```
use strict;
use Getopt::Std ;
my %opts;

getopt('aBc', \%opts) ;
if (defined $opts{'a'}) { print "a: $opts{a}\n"; }
       else { print "a undef\n";}
if (defined $opts{B}) { print "B: $opts{B}\n" ;}
       else { print "B undef\n";}
if (defined $opts{c}) { print "c: $opts{c}\n"; }
       else { print "c undef\n";}
```

Getopt::Std getopts exemple 1

```
use strict;
use Getopt::Std ;
our $opt_a ;
our $opt_B ;
our $opt_c;

getopts('aB:c:') ; # -a est un booléen, -B et -c nécessitent
                   # une valeur associée
if (defined $opt_a) { print "a: $opt_a \n"; } else { print "a undef\n";}
if (defined $opt_B) { print "B: $opt_B \n"; } else { print "B undef\n";}
if (defined $opt_c) { print "c: $opt_c \n"; } else { print "c undef\n";}
```

Getopt::Std getopts exemple 2

Accès aux valeurs des options par %opts

```
use strict;
use Getopt::Std ;
our %opts;          # les options sont dans %opts

getopts('aB:c:', \%opts) ; # -a est un booléen,
             # -B et -c nécessitent une valeur associée

if (defined $opts{'a'}) { print "a: ".$opts{'a'}."\n"; }
else { print "a undef\n";}
if (defined $opts{'B'}) { print "B: ".$opts{'B'}."\n"; }
else { print "B undef\n";}
if (defined $opts{'c'}) { print "C: ".$opts{'c'}."\n"; }
else { print "c undef\n";}
```

Getopt::Std Documentation

- La documentation complète sur le module `Getopt::Std` est disponible en ligne ici :
  https://perldoc.perl.org/Getopt/Std.html

Getopt::Long

Options :

- Composées d'un ou de plusieurs caractères
- Introduites par un ou deux -
- Définies à l'aide de la fonction GetOptions
- Valeurs :
  - Booléennes (définies ou non)
  - Quelconques : chaines de caractères, tableaux ou tables de hachage
  - Possibilité d'associer plusieurs valeurs à une option

Getopt::Long

- Spécification du comportement associé à l'option (caractère qui suit l'option) :
  - `!`: possibilité de spécifier l'option ou sa valeur opposée

    `positive!`

    options possibles :

    ```
    --positive      # valeur 1
    --nopositive    # valeur 0
    ```
  - `+`: incrémentation de la valeur de l'option

    `counter+`

    `--` `counter` (incrémentation de la valeur de la variable associée)

Getopt::Long

- Association d'options courtes et longues à une même variable : |

  `taille|longueur|l|s`

- Spécifiation du type de l'option : caractère qui suit = (valeur obligatoire) ou : (valeur facultative)
  - i : entier
  - f : réel
  - s : chaine de caractères
  - o : entier représenté suivant les conventions Perl (0x9, 0777, 1b)

  Exemples :

  ```
  taille=i
  file:s
  ```

Getopt::Long Exemple 1

```
use Getopt::Long;

my $help;
my $verbose;
my @files;
my %opts;
my $opt;

GetOptions('help|h|?' => \$help, 'verbose!' => \$verbose,
                        'file|f=s' => \@files, 'opt|o=s' => \%opts) or die "Probleme\n";
if (defined $help) {
        print "Option help definie: $help\n";
}
if (defined $verbose) {
        print "Option verbose definie: $verbose\n";
}
if (scalar(@files) > 0) {
        print "Option file definie: ".join(':',@files)."\n";
}
if (scalar(keys %opts) > 0) {
        print "Option opts definie:\n";
        for $opt (keys %opts) {
                print "\t$opt: ".$opts{$opt}."\n";
        }
}

# perl testoptlong.pl --help -f  truc -f machin -o essai=3 -o vitesse=100
```

Getopt::Long Exemple 2 (projet)

```
#!/usr/bin/perl -w
use Getopt::Long;

GetOptions(\%options,
           "verbose", "debug", "commit", "help|?");

if (scalar(keys %options) > 0) {
        print "Options definies:\n";
        for $opt (keys %options) {
                print "\t$opt: est definie\n";
        }
}
if ($options{'help'}) {
  print "Usage: $0 [--verbose --debug --commit|-c --help|?]\n";
  print " ";
  print "Synchronise les utilisateurs et les groupes depuis les donnees du SI\n";
  print "Options\n";
  print "  --verbose|-v                  mode bavard\n";
  print "  --debug|-d                    mode debug\n";
  print "  --commit|-c                   applique les changements\n";
  print "  --help|-h|-?                  affiche ce message d'aide\n";
  exit (1);
}
```

Getopt::Long

- Le module `Getopt::Long` offre de nombreuses possibilités pour l'expression des options d'une application Perl
- Par exemple il et possible d'accepter de regrouper des options booléennes et écrire `-vax` au lieu de `-v -a -x`
- Pour pouvoir le faire il faut appeler la méthode `Getopt::Long::Configure ("bundling");`
- Une documentation très complète sur Getopt::Long est disponible ici :

  https://perldoc.perl.org/Getopt/Long.html

Utilisation d'une base de données MySQL

Module Perl DBI

Connexion à la Base de Données

```
#!/usr/bin/perl
use strict;

use DBI();
# configuration de la base de données MYSQL
my $dsn = "DBI:mysql:database=si;host=sql.imss.org";
my $username = "root";
my $password = 'dbmaster';

# connexion à la BD MySQL
my %attr = ( PrintError=>0,  # turn off error reporting via warn()
             RaiseError=>1}; # turn on error reporting via die()

my $dbh  = DBI->connect($dsn,$username,$password, \%attr);

# on est connecté, $dbh est le HANDLE de la base de données

# en fin d'utilisation de la BD faire
$dbh->disconnect;
```

Insertion dans la BD

- Pour insérer une nouvelle ligne dans une table à l'aide de Perl DBI, procédez comme suit:
  - Vérifier l'instruction INSERT en appelant la méthode prepare() de l'objet handle de base de données.
  - La méthode prepare () renvoie un objet descripteur d'instruction qui représente une instruction dans la base de données MySQL. Dans cette étape, Perl DBI valide l'instruction INSERT pour s'assurer de sa validité. S'il existe une erreur dans l'instruction INSERT, par exemple si elle fait référence à une table inexistante ou s'il s'agit d'une instruction SQL non valide, l'instruction prepare () renvoie la valeur undef. De plus, Perl renseigne le message d'erreur dans la variable $DBI::errstr.

Insertion dans la BD

- Exécutez l'instruction INSERT en appelant la méthode execute()de l'objet descripteur d'instruction. À cette étape, Perl exécute l'instruction INSERT dans la base de données MySQL. La méthode execute () renvoie true en cas de succès et la valeur undef en cas d'échec. En cas d'échec, Perl déclenche également une exception via la fonction die () pour abandonner le script immédiatement si l'attribut RaiseError est activé.
- Notez que vous pouvez exécuter les instruction UPDATE ou DELETE en utilisant ces étapes.

Insertion dans la BD

```
# insertion de données dans la table groupes
my $sql = "INSERT INTO groupes(id_groupe,nom_groupe,description)
    VALUES(?,?,?)";

my $stmt = $dbh->prepare($sql);

# execution de la requête
$stmt->execute(10001,'groupe1','le premier groupe');
```

Transaction

- Perl DBI fournit un ensemble d'API qui vous permet de traiter efficacement les transactions. Pour gérer les transactions dans Perl DBI, procédez comme suit :
  - Donnez la valeur false à l'attribut AutoCommit pour activer la transaction.
  - Exécutez des opérations dans un bloc eval.
  - À la fin du bloc eval, appelez la méthode commit () de l'objet descripteur de base de données pour valider les modifications.
  - Recherchez l'erreur dans la variable $@ et appelez la méthode rollback () de la base de données pour annuler les modifications si une erreur se produit.

Transaction

```
# connexion à la BD
my %attr = ( PrintError=>0,  # turn off error reporting via warn()
             RaiseError=>1};   # turn on error reporting via die()
my $dbh  = DBI->connect($dsn,$username,$password, \%attr);
# on est connecté, $dbh est le HANDLE de la base de données
eval{
# insertion de données dans la table groupes
 my $sql = "INSERT INTO groupes(id_groupe,nom_groupe,description)
   VALUES(?,?,?)";
 my $stmt = $dbh->prepare($sql);

 # execution de la requête
 $stmt->execute(10001,'groupe1','le premier groupe');

 # si tout est ok, on confirme à la base de données
 $dbh->commit();
};
if($@){
 print "Erreur à l'insertion: $@\n";
 $dbh->rollback();
}
```

Interrogation de la BD

Pour interroger la base de données MySQL depuis un programme Perl, procédez comme suit :

- Préparez une instruction SELECT pour l'exécution à l'aide de la méthode prepare() de l'objet handle de base de données.
- La méthode prepare() renvoie un objet descripteur d'instruction qui représente une instruction dans la base de données MySQL.
- Exécutez l'instruction SELECT en appelant la méthode execute() de l'objet descripteur d'instruction.
- Appelez la méthode fetchrow_array(), fetchrow_arrayref() ou fetchrow_hashref () pour extraire des données du jeu de résultats jusqu'à ce qu'il ne reste plus de ligne. Vous pouvez utiliser l'instruction while pour itérer le jeu de résultats.

Interrogation de la BD

```
# recuperation des utilisateurs du SI
$query = "SELECT nom, prenom,identifiant,num_utilisateur,mot_passe,courriel,
date_expiration FROM utilisateurs"  ;
$sth = $dbh->prepare($query);
# execution de la requête
$row = $sth->execute;

while ($row = $sth->fetchrow_hashref) {
   $user = $row->{identifiant};
   push(@SIusers,$row->{identifiant});
   printf "%s %s %s %s %s\n", $row->{identifiant}, $row->{nom},
          $row->{prenom}, $row->{courriel}, $row->{id_utilisateur};
}
```

Mise à jour de la BD avec UPDATE

Pour mettre à jour des données dans une table, vous utilisez l'instruction UPDATE.

- construisez une instruction UPDATE et transmettez-la à la méthode prepare () de l'objet descripteur de base de données pour préparer la requête à exécuter
- si vous souhaitez transmettre des valeurs du programme Perl à l'instruction UPDATE, vous devez placer des '?' à la place des valeurs dans l'instruction, et lier le paramètre correspondant à l'aide de la méthode bind_param () de l'objet descripteur d'instruction.
- Après cela, appelez la méthode execute() de l'objet descripteur d'instruction pour exécuter la requête.

Mise à jour de la BD

```
# preparation de la requête UPDATE
my $sql = "UPDATE nom_table SET nom_champ = ? WHERE id=?";
my $sth = $dbh->prepare($sql);

# definition de la nouvelle valeur du(des) champ(s) à modifier
$sth->bind_param(1,$value);
$sth->bind_param(2,$id);

# execution de la requête
$sth->execute();
```