Introduction à *Perl*

Jean-Michel Richer
jean-michel.richer@univ-angers.fr
http://www.info.univ-angers.fr/pub/richer

Université d'Angers
LERIA - Laboratoire d'Etudes et de Recherche
en Informatique d'Angers

2008

Outline

Introduction

Définition

Perl est un langage de script, généraliste, multiplateforme, libre (Open Source), gratuit né du besoin de disposer d'un langage optimisé pour l'extraction d'information dans des fichiers texte ainsi que la génération de rapports.

Si on ne dispose pas de Perl, on peut utiliser dans une moindre mesure le Shell ainsi que les différents programmes associés : `sed`, `awk`, `grep`, `cut`, `test` et `expr`.

PERL ?

- **Perl** fut créé par Larry Wall en 1987 reprenant les fonctionnalités du langage C et de programmes comme sed, awk ou le shell sh.
- Larry Wall donne deux interprétations de l'acronyme :
  - Practical Extraction and Report Language
  - Pathetically Eclectic Rubbish Lister

LISP : List Processing $\Rightarrow$ Lots of Insipide and Stupid Parenthesis

Caractéristiques du langage

Les caractéristiques de *Perl* sont les suivantes :

- volonté de produire un langage proche du langage naturel
- plusieurs possibilités d'exprimer le même traitement (TIMTOWTDI = *There is more than one way to do it*)
- langage typé dynamiquement (voire statiquement)
- traitement des expressions régulières (recherche, substitution de texte)
- nombreuses bibliothèques (libraries)
- interopérabilité avec d'autres langages (Langage C)

Dans quel cas Utiliser *Perl*

Perl est utilisé pour extraire de l'information de fichiers texte, l'organiser et la traiter :

- développement Web (Serveur Apache)
- gestion électronique de document
- bioinformatique (bases de données, fichiers PDB, bioperl)
- tout traitement sur des fichiers texte afin d'extraire de l'information

Documentation et Aide

- site officiel : `www.perl.org`
- documentation CPAN (Comprehensive Perl Archive Network) : `cpan.perl.org`
- installation de la documentation en local : perldoc
- *Perl* en français : `perl.enstimac.fr`
- *Perl* Mongers (groupes d'utilisateurs) : `www.pm.org`

Ecrire un programme *Perl*

Pour créer un programme *Perl*, il faut écrire un script (fichier texte) d'extension `.pl` et qui commence par la ligne :

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

On suppose ici que le programme `perl` se trouve dans le répertoire `/usr/bin` sous Linux/Unix.
On donnera le droit en exécution au programme :

```
chmod +x mon-programme.pl
```

Les types de données

Il existe 3 types principaux de structures de données en *Perl*

- les scalaires
- les tableaux
- les structures de hachage (mapping en anglais) également appelées tableaux associatifs

on peut également ajouter :

- les descripteurs de fichiers

Les scalaires

Scalaire

Les valeurs scalaires (ou atomiques) sont préfixées par le symbole $. Il s'agit des entiers, des réels et des chaînes de caractères.

Exemples :

- `$entier = 1970;`
- `$reel = -12.78E+3;`
- `$chaine = "dessine moi un mouton";`

Opérateur des entiers

Les opérateurs associés aux entiers sont :

symbole	opérateur
+	addition
-	soustraction
/	division entière
*	multiplication
%	modulo (reste de la division entière)
++	incrémentation (post, pré)
- -	décrémentation (post, pré)

Opérateur des réels

Les opérateurs associés aux réels sont :

symbole	opérateur
+	addition
-	soustraction
/	division
*	multiplication
**	puissance

Définition des chaînes de caractères

Les chaînes de caractères peuvent être introduites :

- soit par le caractère ’
- soit par le caractère " qui permet une interpolation (interprétation de des variables)

Par exemple :

```
@a=(1,2,3);
$b=12.56;
$chaine = "@a $b";

print $chaine, "\n";
```

affichera : *1 2 3 12.56*

Opérateur des chaînes de caractères

Les opérateurs associés aux chaînes de caractères sont :

symbole	opérateur
.	concaténation
x	duplication
+	addition (si nombres)

Exemples :

- `$c = $a . $b;`
- `$c = $a x 20;`
- `$a = '20'; $b = '10'; $c = $a + $b;`

Fonctions associées aux chaînes de caractères

Fonction	rôle
`chop`	supprime le dernier caractère
`chomp`	supprime les caractères de passage à la ligne
`eval`	évalue le code PERL contenu dans la chaîne
`length`	longueur de la chaîne
`lc`	passage en minuscule
`reverse`	inverse
`substr`	obtenir une sous-chaîne
`uc`	passage en majuscule

Caractères spéciaux

caratère	description
\n	passage à la ligne (LF)
\r	retour chariot (CR)
\t	tabulation
\u	afficher en majuscule le caractère suivant
\U	afficher en majuscule tous les caractères qui suivent
\l	afficher en minuscule le caractère suivant
\L	afficher en minuscule tous les caractères qui suivent
\E	fin de \U ou \L

Fonctions associées aux chaînes de caractères

```
$chaine = "le chat mange la souris";
@mots=split(/ /,$chaine);
print join(',',@mots), "\n";
$tmp = substr($chaine,3,4);
print "$tmp\n"; # affiche 'chat'
$tmp2 = reverse($tmp);
print "$tmp2\n";
print scalar reverse($tmp), "\n"; #!!
warning!!
```

Affichage d'information

Pour afficher de l'information on utilise l'instruction `print` :

```
$a=1; $b=-56.367; $c='toto';
print $a, ' ', $b, ' ', $c, "\n";
print "$a $b $c\n";
```

le résultat de l'affichage est le même dans les 2 cas :

```
1 -56.367 toto
```

Affichage d'information formatée

On utilise la fonction `sprintf` pour formater l'information :

- % introduit un nouvelle valeur à afficher suivie du format d'affichage
- s correspond à une chaîne de caractères
- d correspond à un entier
- f correspond à un réel

```
print sprintf("%10s %04d %.2f", 'toto',
3, 1133.568);
```

affichera : `toto 0003 1133.57`

Les tableaux

Tableaux

Les noms de tableaux sont préfixés par le symbole @. On peut accèder aux éléments par leur indice qui commence à 0.

Exemples :

- `@noms = ('jean', 'paul', 'pierre');`
- `@nombres = (1,3,5,7,9,11);`
- `@mille = (1..1000);`
- `print $noms[1]; #!! warning!!`

Fonctions de manipulation des tableaux

fonction	description
`scalar`	nombre d'éléments
`push`	ajouter un élément à la fin du tableau
`unshift`	ajouter un élément au début du tableau
`pop`	supprime le dernier élément
`shift`	supprime le premier élément
`split`	transformer une chaîne en tableau
`join`	transformer un tableau en chaîne
`sort`	tri

Autres fonctions associées aux tableaux

- qw (quote word) transforme des mots un en tableau de chaînes
- on peut également récupérer une partie des éléments du tableau sous la forme suivante :

```
@tab=(1,2,3,4);
($a,$b)=@tab;

print "$a $b\n";
```

affichera : `1 2`

Exemple de manipulation de tableau

Exemple :

```
@array=qw/le chat mange la souris/;

print "@array\n";
print scalar @array, "\n";

$dernier=pop @array;
push @array, "petite";
push @array, $dernier;

print "@array\n";
```

Les structures de hachage

Hachage / Mapping

Les structures de hachage permettent d'associer une clé et une valeur. Elles sont préfixés par le symbole %. On peut accèder à une valeur par l'intermédiaire de sa clé.

Exemples :

- `%colors = (rouge => 1, vert => 2, bleu => 3);`
- `%colors = ('rouge',1,'vert',2,'bleu',3);`
- `print $colors{rouge};`
- `print $colors{'vert'};`

Fonctions associées aux mappings

fonction	description
`keys`	donne la liste des clés
`values`	donne la liste des valeurs
`each`	donne clé et valeur (à utiliser avec while)

Exemple de manipulation de mapping

Exemple :

```
%notes = ( francais => 12, anglais =>
14, maths => 17);
$somme=0;
foreach $cle (keys %notes) {
  $somme = $somme + $notes{$cle};
}
# ou
while (($cle,$note)=each(%notes)) {
  $somme = $somme + $note;
}
$moyenne = $somme / scalar keys %notes;
print "$moyenne\n";
```

Les structures de contrôle

Les structures de contrôle représentent les différents éléments syntaxiques qui permettent de structurer le programme afin de concevoir les traitements sur les données. On distingue :

- la conditionnelle : `if-then-elsif-else`
- les structures de boucles :
  - `for`
  - `foreach`
  - `while`

La conditionnelle : `if-then-elsif-else`

La structure du `if` est la suivante :

Conditionnelle

```
if (condition_1) {
   instructions_1;
} elsif (condition_2) {
   instructions_2;
} else {
   instructions_3;
}
```

Opérateurs logiques

Les opérateurs logiques et de comparaison sont utilisés dans l'expression de la condition du `if`.
Les opérateurs logiques sont :

symbole	symbole	opérateur
`||`	`or`	ou logique
`&&`	`and`	et logique
`!`	`not`	négation
	`xor`	ou exclusif

Opérateurs de comparaison

Les opérateurs de comparaison sont :

symbole	symbole	opérateur
==	eq	égal
!=	ne	différent
<	lt	inférieur
<=	le	inférieur ou égal
>	gt	supérieur
>=	ge	supérieur ou égal

La conditionnelle : `if-then-elsif-else`

Exemple :

```
if ($x == 1) {
   print "x vaut 1";
}
```

Exemple :

```
if (($x % 2)==0) {
   print "$x est un nombre pair\n";
} else {
   print "$x est un nombre impair\n";
}
```

La structure de boucle **for**

pour

```
for (initialisation; condition; incrementation)
{
  instructions;
}
```

- *initialisation* : des variables de la boucle
- *condition* : condition d'arrêt
- *incrémentation* : des variables de la boucle

La structure de boucle **for**

Exemple : affichage des valeurs entières de i=1 à 10.

```
for ($i=1;$i<=10;++$i) {
   print "i = $i\n";
}
```

La structure de boucle **foreach**

pour chaque

```
foreach variable (liste) {
   instructions;
}
```

La structure de boucle **foreach**

Exemple : affichage des valeurs entières de i=1 à 10.

```
foreach $i (1..10) {
   print "$i\n";
}
```

La structure de boucle **while**

Tant que

```
while (condition) {
   instructions;
}
```

On exécute les instructions tant que la condition est vérifiée.

La structure de boucle **while**

Exemple : affichage des valeurs entières de i=1 à 10.

```
$i=1;
while ($i<=10) {
   print "i = $i\n";
   ++$i;
}
```

Définition des fonctions

Sous-programme

Un sous-programme (fonction ou procédure) permet de regrouper un ensemble de traitements qui pourront par la suite être appliqués à des données différentes. On introduit un sous-programme grâce au symbole `sub`.

- une fonction est un sous-programme qui retourne une valeur
- une procédure n'a pas de valeur de retour
- à l'intérieur du sous-programme, le tableau `@_` représente l'ensemble des paramètres passés au sous-programme lors de son appel.

Variables locales et globales

Local ou glogal

- les variables **locales** sont introduites grâce au symbole `my`
- les variables **globales** sont introduites grâce au symbole `our`

Une variable est dite locale au sous-programme si elle n'est utilisée qu'à l'intérieur du sous-programme.

Exemple de sous-programme

Exemple : somme de deux nombres

```
sub somme {
   my ($a,$b)=@_;
   return $a + $b;
}

print somme(1,2), "\n";
print somme(1.25,2.56), "\n";
print somme('1','2'), "\n";
```

Les handle de fichiers

Handle de fichier

On utilise un descripteur (appelé handle) pour manipuler un fichier. Pour ouvrir un fichier on utilise la commande open.

Handle par défaut :

- `STDIN` (flux d'entrée standard) correspond au clavier
- `STDOUT` (flux de sortie standard) correspond à l'écran
- `STDERR` (flux d'erreur standard) correspond à l'écran

Ouverture et fermeture des fichiers

```
$nom_fichier="notes.txt";

open(FILE,"<" . $nom_fichier) or
die("impossible d'ouvrir
$nom_fichier\n");
close(FILE)
```

Exemples :

- `open(FILE,'<toto.txt');` ouverture en lecture
- `open(FILE,'>toto.txt');` ouverture en écriture
- `open(FILE,'>>toto.txt');` ouverture en ajout

Lecture des fichiers

lecture ligne à ligne :

```
while ($ligne=<FILE>) {
   print $ligne;
}
```

ou plus rapidement :

```
@lines=<FILE>;
print @lines;
```

Qu'est ce qu'une expression régulière ?

Expression régulière

Une expression régulière (ER) ou motif est une expression qui permet de représenter en ensemble de chaînes de caractères. Elles permettent de reconnaître et d'isoler des parties d'un texte.

Par exemple : recherche de l'ensemble des liens d'une page HTML.

Opérateurs liés aux expressions régulières

Il existe 3 opérateurs pour les ER :

- `m/motif/` qui permet de rechercher un motif
- `s/motif1/motif2/` qui permet de substituer un motif1 par motif2
- `tr/set1/set2/` qui permet de remplacer un ensemble de caratères par un autre ensemble

On utilise en particulier l'opérateur `=~` pour appliquer une expression régulière à une chaîne.

Exemple

```
$chaine = "le chat mange la souris";
$chaine =~ s/chat/matou/;
print "$chaine\n";
$chaine =~ tr/aeiou/AEIOU/;
print "$chaine\n";
```

affiche :

```
le matou mange la souris
lE mAtOU mAngE lA sOUrIs
```

Description des expressions régulières

On donne un sens particulier à certains caractères :

symbole	description
.	n'importe quel caractère sauf le saut de ligne
|	alternative : expression de gauche ou à droite
( )	regroupement
[ ]	classe de caractères
^	début de chaîne
$	fin de chaîne

Description des expressions régulières

On peut indiquer une répétition :

symbole	description
`*`	0 ou plus
`+`	1 ou plus
`?`	0 ou une fois
`{n}`	*n* fois exactement
`{n,}`	au moins *n* fois
`{n,m}`	entre *n* et *m* fois

Description des expressions régulières

Certains caractères remplacent avantageusement un ensemble de caractères :

symbole	description
\s	espace, tabulation , CR, LF
\S	tout sauf un espace, tabulation, CR, LF
\w	lettre ou chiffre
\W	tout sauf lettre et chiffre
\d	tout chiffre
\D	tout caractère non chiffre

Description des expressions régulières

Les modificateurs peuvent être ajoutés à la fin de l'expression (après le dernier /) de manière à en modifier le sens :

symbole	description
\i	ignore la distinction majuscule, minuscule
\s	. englobe \n
\m	^ et $ associés à \n
\x	ignore les blancs et autorise les commentaires
\o	ne compile qu'une seule fois le motif
\g	recherche globale pour détecter toutes les occurrences
\cg	permet de continuer la recherche après un échec de /g

Exemples

Vérification d'adresse email :

```
$chaine = "jm-richer\@univ-angers.fr";
if ($chaine =~
m/([\w|\-]+)\100([\w|\-]+).fr/) {
   print "adresse email valide\n";
}
```

Recherche d'occurrences :

```
$chaine="Perl, la perle des perles";
if (@perls = $chaine =~ m/perl/gi) {
   printf "trouve %d fois", scalar
@perls;
}
```

Conclusion

Finalement, on peut dire que :

- *Perl* est un langage très puissant
- mais quelque peu complexe à apprendre pour le néophyte
- il faut pratiquer régulièrement (programmer) si on désire maîtriser Perl
- d'autres langages de script assez proches de *Perl* existent : python et php

Quelques trucs et astuces

- pour modifier `chaine1` en `chaine2` dans tous les fichier d'extension `.sh` du répertoire courant :
  `perl -pi -e 's/chaine1/chaine2/go;' *.sh`
- pour changer les dates de la forme 30/09/70 en 1970-09-30 dans tous les fichiers du répertoire courant :
  `perl -pi -e 's|(\d+)/(\d+)(\d+)|19$3-$1-$2|go;' *`
- renommer tous les fichiers `fichier.f` en `fichier.ff` :
  `perl -e 'foreach (<*>) { rename("$1.f","$1.ff") if (/(.*).f$/);}'`

Bibliographie

- Programmation en Perl, *Larry Wall, Tom Christiansen and Jon Orwant*, O'Reilly, 2001
- Introduction à Perl pour la bioinformatique, *James Tisdall*, O'Reilly, 2002