Introduction à Perl 4 et 5

Magali Contensin

Table des matières

1 Introduction

Perl (*Practical Extraction and Report Language*) a été créé par Larry Wall en 1986.

Caractéristiques

- langage de programmation interprété :
  - pas de compilation ;
  - il faut perl sur l'ordinateur pour exécuter les scripts ;
  - plus lent qu'un programme compilé (ne pas faire de calculs) ;
  - plus rapide que le Bourne Shell ;
- aucune allocation mémoire à gérer ;
- existe pour plusieurs plateformes Unix, Linux, Windows, Mac ;
- gratuit (téléchargeable sur `http://www.perl.com`) ;
- pour les interfaces interactives : tkperl (pas portable);
- perl 5 intègre la programmation objet.

Utilisation

- gestion des expressions régulières ;
- manipulation de fichiers ;
- manipulation de textes ;
- manipulation de processus ;
- génération de fichiers html (CGI) ;
- accès aux bases de données ;
- conversions de formats de fichiers.

Compilation

Vous pouvez compiler votre fichier en appelant perl sur la console :

1. créer un fichier qui contient : `print "\nBonjour\n";`
2. lui donner l'extension `.pl` ;
3. taper sur la console `perl monfichier.pl` ;
4. vous devriez obtenir `Bonjour`

Mais vous pouvez aussi faire un script qui indique au shell qu'il doit lancer le programme dont le nom suit le #! et avec les paramètres p1, p2, ..., pn :

1. créer un fichier qui contient :

```
#!/usr/bin/perl -p1 -p2 -pn
print "\nBonjour\n";
```

2. lui donner l'extension `.pl` ;
3. faire de ce script un exécutable : `chmod +x monfichier.pl`
4. taper sur la console `monfichier.pl` ;
5. vous devriez obtenir aussi `Bonjour`

Passage de paramètres

Les paramètres passés par la ligne de commande sont stockés dans le tableau `@ARGV` et pour accéder aux paramètres on utilise `$ARGV[n]`.

Le script ci-dessous affiche tous les paramètres passés par la ligne de commande.

```
#!/usr/bin/perl
foreach $param (@ARGV){
    print "$param ";
}
```

2 Données

2.1 Variables

syntaxe : `$ma_variable = donnee;`

```
#!/usr/bin/perl

$reel = 0.3; print $reel ." , ";
$reel2 = 0.1e+2; print $reel2 . " , ";
$entier = 22; print $entier . " , ";
$octal = 0377; print $octal. " , ";
```

```
$hexadecimal = 0xff; print $hexadecimal;

$sport = 'ski';
$chaine = 'j\'aime le '. $sport;
print "\n$chaine\n";
print "j\'aime le $sport\n";
print 'j\'aime le $sport\n';

$chaine = <<_web;
<HTML>
<HEAD>
   <TITLE> titre page </TITLE>
</HEAD>
<BODY>
   voici le corps de ma page
</BODY>
</HTML>
_web
print "\n".$chaine;
```

Variables particulières

`$_`	dernière ligne lue (dans while)
`$!`	dernière erreur
`$$`	numéro système du programme en cours
`#1,...,#n`	contenu de la parenthese n dans l'expression régulière
`#0`	nom du programme

2.2 Tableaux

2.2.1 Tableau simple

syntaxe : `@montableau = (elt1, elt2, ..., eltn);`

```
@nombrepremiers = (1, 3, 5, 7, 11, 13, 17, 23);
@chiffre = (0..9);
@alphabet = ('a'..'z');
```

Référence de 0 à n-1 : `$montableau[elt]`
Dernier indice donné par `$#montableau` (taille - 1)

```
print $nombrepremiers[4]."\n";
print $#nombrepremiers."\n";
print $chiffre[4]."\n";
print $#chiffre."\n";
print $alphabet[4]."\n";
print $#alphabet."\n";
```

2.2.2 Tableau indicé

syntaxe : `%montableau = (elt1=>donnee2, elt2=>donnee2, ..., eltn=>donneen);`

```
%alpha = (1=>'un', '2'=>'deux', 3=>'trois');
```

Référence de 0 à n-1 : `$montableau{elt}` fournit `donnee`

```
print "1 = $alpha{1}\n";
print "2 = $alpha{2}\n"
```

2.2.3 Tableau de tableau

```
%saison = (
    'ete'=>['juin','juillet','aout','septembre'],
    'automne'=>['septembre','octobre','novembre','decembre'],
    'hiver'=>['decembre','janvier','fevrier','mars'],
    'printemps'=>['mars','avril','mai','juin']);
```

print `$saison{'ete'}[0];` donnera 'juin'

Tableaux particuliers

`@_`	contient les paramètres passés à une fonction
`@ARGV`	contient les paramètres passés au programme
`%ENV`	tableau indicé qui contient les variables d'environnement

3 Opérateurs

3.1 Arithmétiques

+	addition
-	soustraction
*	multiplication
/	division
**	exponentielle
%	modulo
++	incrémentation
--	décrémentation

3.2 Booléens

`&&` ou and	et logique
`||` ou or	ou logique
! ou not	négation

3.3 Affectation

=	affectation
x += y	x = x + y
x -= y	x = x - y
x *= y	x = x * y
x /= y	x = x / y
x **= y	x = x ** y
x %= y	x = x % y

3.4 Comparaison

Opérateur	Numérique	Chaîne
égal	==	eq
différent	!=	ne
inférieur	<	lt
supérieur	>	gt
inférieur ou égal	<=	le
supérieur ou égal	>=	ge

3.5 Chaines de caractères

.	concaténation
x	réplique

```
#!/usr/bin/perl
$c1 = "neige ";
$c2 = "bonhomme de ".$c1;
print $c2;
$c2 x= 3;
print $c2;
```

4 Instructions

4.1 if

```
if (expression){
    instruction1;
    ...
}
else {
    instruction1
    ...
}
```

Exemple :

```
print "\n entrez un nombre : ";
$x = <STDIN>;
```

```
print "et un autre nombre  : ";
$y = <STDIN>;

if($x == $y){
    print "egal";
}elsif ($x > $y){
    print "superieur";
}else
{
    print "inferieur";
}
```

négation : commande `unless`

```
unless ($x == $y){
    print "\ndifferents";
}
```

4.2 while

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

Exemple :

```
while ($x < $y){
    $x++;
    print "$x ";
}
```

4.3 do ... while

```
do{
    instructions;
}
while (expression)
```

4.4 for et foreach

```
for(expression; test; expression_inc){
    instructions;
}

foreach var (@tableau){
    instructions;
}
```

Exemples :

```
for($i = 1; $i < 6; $i++){
```

```
    print "$i ";
}

@sport = ('ski', 'natation', 'randonnée');
foreach $i (@sport)
{
    print "\t $i \n";
}
```

4.5 goto

`goto label;` va à la ligne référencée par `label :`

4.6 interruption de boucles

`last` et `next`

```
print "\nessai de next : ";
$i = 0;
while($i < 10)
{
    $i++;
    if($i == 5){
        next;
    }
    print "$i ";
}

print "\nessai de last : ";
$i = 0;
while($i < 10)
{
    $i++;
    if($i == 5){
        last;
    }
    print "$i ";
}
```

5 Fonctions

5.1 Syntaxe

```
sub nomfonction{
    instructions ;
}
```

Appel : `&nomfonction;`

```
sub maprocedure{
    print "un essai de fonction\n";
}
&maprocedure;
```

Valeur retournée :

- aucune : par défaut retourne la valeur de la dernière instruction ;
- celle donnée par return.

```
sub fonction1{
    $x = 3*5;
    $y = 2+1;
}
print &fonction1;

sub fonction2{
    return 3*7;
}
print &fonction2;
```

5.2 Arguments

La liste de variables suivant l'appel de la fonction représente les arguments de la fonction. C'est le tableau spécial `@_` qui les stocke, on peut y accéder en écrivant : `$_[n]`.

```
sub maxi{
    if($_[0] < $_[1]){
        return $_[1]
    }
    else{
        return $_[0]
    }
}
print "\n donnez un nombre : "; $x = <STDIN>;
print "\n donnez un nombre : "; $y = <STDIN>;
print "le maximum est ".&maxi($x, $y)."\n";
```

5.3 Variables globales et locales

Dans une fonction les variables déclarées sont globales. Pour indiquer qu'elles sont locales il faut utiliser `local` avant perl version 5 et `my` en perl 5.

```
$x = 5;
sub essai{
   local $x = 6;
}
&essai;
print $x;
```

5.4 Fonctions prédéfinies

5.4.1 Mathématiques

nom	description
abs(x)	valeur absolue
cos(x)	cosinus
exp(x)	exponentielle
int(x)	partie entière
log(x)	logarithme
sin(x)	sinus
sqrt(x)	$\sqrt{(x)}$
rand	entier pseudo-aléatoire
srand	nouvelle séquence de nombres pseudo-aléatoires
tan(x)	tangente

5.4.2 Chaînes de caractères

nom	description
chop(chaine)	suppression du dernier caractère
chomp(chaine)	suppression du retour chariot
index(chaine, sschaine)	indice de la première occurrence de chaine dans sschaine
length(chaine)	taille de la chaine
lc(chaine)	retourne la chaine en minuscules (perl 5)
lcfirst(chaine)	retourne la chaine avec le premier caractère en minuscule
split(sep,chaine)	sépare la chaine en éléments en fonction du séparateur
substr(chaine, début, longueur)	extraction d'une sous-chaine
uc(chaine)	retourne la chaine en majuscules (perl 5)
ucfirst(chaine)	met le premier caractère de la chaine en majuscule

5.4.3 Tableaux

grep(exp,tab)	recherche l'expression dans un tableau
join(sep, tab)	regroupe les éléments de tab dans une chaine en insérant le séparateur
pop(tab)	retourne le dernier élément et le supprime
push(tab,elt)	insère des éléments en fin de tableau
reverse(tab)	inverse l'ordre des éléments
shift(tab)	retourner et supprimer le premier élément
sort(tab)	trie par ordre croissant les éléments.
splice(tab, deb, n)	enlève n éléments de tab en commençant à deb
unshift(tab, elt)	insère des éléments en début de tableau

```
@sport = ('ski', 'natation', 'randonnée'); print " @sport";
shift(@sport); print "\n @sport";
unshift(@sport, 'tennis'); print "\n @sport";
@sport = sort(@sport); print "\n @sport";
@sport = reverse(@sport); print "\n @sport";
pop(@sport); print "\n @sport";
push(@sport,'pelote-basque');
push(@sport,'ping-pong'); print "\n @sport";
```

```
@supprime = splice(@sport, 2, 2);
print "\n @sport";
print "\n @supprime \n";
```

5.4.4 Tableaux indicés

keys(tab)	liste des clés
values(tab)	liste des valeurs
each(tab)	parcourt la liste et donne clé-valeur
delete	supprime une clé-valeur de la liste
exists	pour savoir si un élément a été défini

```
@listecle = keys(%alpha); print "\n@listecle";
@listevaleurs = values(%alpha); print "\n@listevaleurs";
```

5.4.5 Fichiers

nom	description
chmod	changer les droits d'accès
chown	changer le propriétaire
eof	détection de la fin du fichier
mkdir	création de répertoire
rename	renommer le fichier
rmdir	suppression de répertoire

5.4.6 Système

nom	description
chdir	changer le répertoire de travail
die	interruption du script et affichage d'un message
exec	exécution d'un programme
exit	interruption du script
getlogin	donne le nom du login
sleep	suspend l'exécution pendant une durée
system	exécution d'un programme externe, le script est suspendu

6 Fichiers

Accès aux fichiers textes par des pointeurs de fichiers référencés par des variables.

6.1 Ouverture

```
open(NOMPOINTEUR, "(prefixe)nomFichier");
```

préfixe	description
<	lecture (par défaut)
>	écriture (fichier écrasé)
>>	ajout
|	résultat envoyé à un programme

Les pointeurs de fichiers STDIN, STDOUT et STDERR sont déjà définis, ils ne faut pas les utiliser dans open().

6.2 Fermeture

```
close NOMPOINTEUR;
```

6.3 Lecture et écriture

Accès au contenu du fichier avec <et >.
Écriture avec : `printf NOMPOINTEUR "format", $chaine1, ...;`
Exemple d'un script qui compte le nombre de lignes

```
#!/usr/bin/perl
open(F, 'monfichier.tex') || die "Problème à l\'ouverture : $!";
$i = 0;
while(<F>)
{
    $i ++;
}
print "\nNombre de lignes : $i\n";
close F || die "Problème à la fermeture : $!";
```

6.4 Communication (pipes)

open permet aussi d'exécuter des commandes shell.

```
#!/usr/bin/perl
open(MAIL, "|mail magali");
print MAIL "mon texte de mail\n";
close MAIL;
```

6.5 Accès aux fichiers

-e	vérifie l'existence du fichier ou répertoire
-r	vérifie l'existence et les droits d'accès
-w	on peut écrire dedans
-x	on peut exécuter
-d	c'est un répertoire
-A	date du dernier accès
-M	date de la dernière modification

```
#!/usr/bin/perl
print "donnez un nom de fichier : ";
$chaine=<STDIN>;
chop($chaine);

if (-e $chaine){
     print "le fichier existe\n";
```

```
 if (-r $chaine){
     print "écriture autorisée";
 }
 else{
     print "en lecture seulement";
 }
}
else{
    print "le fichier n'existe pas";
}
```

7 Expressions régulières

L'expression régulière est un modèle à comparer avec une chaîne de caractères (*pattern matching*). Utile pour la recherche, la modification et l'extraction.

Syntaxe d'une expression régulière : `/expression/`

Deux opérateurs :

`=~` recherche d'une correspondance de l'expression à droite sur le paramètre de gauche ;
`!~` négation de `=~`.

.	tout caractère (sauf le \n)
[...]	chercher un des caractères entre crochets
[^...]	tout caractère qui n'est pas dans les crochets
\n \r \f \t	nouvelle ligne, retour chariot, saut de page, tabulation
\w	caractères alphanumériques (A..Z, a..z, 0..9)
\W	tout sauf des caractères alphanumériques
\s	espace
\S	pas d'espace
\A	début de la chaine
\Z	fin de la chaine
\d	chiffre (0..9)
\D	tout sauf un chiffre (0..9)
^e	la ligne (ou la chaine) commence par le caractère e
e$	la ligne (ou la chaine) finit par le caractère e
e+	une ou plusieurs occurrences de e
e*	zéro ou plusieurs occurrences de e
e?	zéro ou une occurrence de e
e{n}	n instances de e
e{n,}	au moins n occurrences de e
e{n,p}	au moins n occurrences de e et au plus p
|	alternative

Pour obtenir les caractères spéciaux :

```
+ ? . * ^ $ ( ) [ ] { } | \
```

il faut les faire précéder de \

7.1 Recherche sans modification

Syntaxe	`m/expression/options`
Options	g : toutes les occurrences ; i : ne pas tenir compte de la casse.

Voici un script qui vérifie les adresses mail passées en paramètre :

```
#!/usr/bin/perl

foreach $param (@ARGV){
   if($param =~ m/[_a-zA-Z\d\-\.]+@[_a-zA-Z\d\-]+(\.[_a-zA-Z\d\-]+)/){
       print "$param\n";
   }
   else{
       print "mauvaise adresse\n";
   }
}
```

Pour un fichier contenant les noms : INXS, Jackson, Nirvana, Police, Sting, Supertramp le script ci-dessous n'affichera pas INXS et Nirvana.

```
#!/usr/bin/perl
print "entrez un nom de fichier : ";
$fichier = <STDIN>;
chomp($fichier);
open(F, $fichier) || die "Problème à l\'ouverture : $!";
while($ligne = <F>)
{
    if($ligne !~ m/N/){
        print $ligne;
    }
}
```

7.2 Recherche avec modification

Syntaxe	`s/expression/remplacement/options`
Options	g : toutes les occurrences ; i : ne pas tenir compte de la casse.

Le script ci-dessous affichera poules ou chevaux si le mot entré était poule ou cheval (le mot entré est modifié dans la variable nom).

```
#!/usr/bin/perl
print "entrez un nom d'animal : ";
$nom = <STDIN>;
chomp($nom);
$nom =~ s/al$/aux/;
$nom =~ s/e$/es/;
print "pluriel : $nom\n";
```

Voici un script qui donne le nombre d'occurrences remplacées, essayez le avec la phrase ci-dessous : *J'aime Internet explorer et j'utilise souvent internet Explorer pour faire des pages web.*

```
#!/usr/bin/perl
print "entrez une phrase : ";
$phrase = <STDIN>;
$nb = ($phrase =~ s/Internet Explorer/Netscape/gi);
print "\n\n$phrase\n";
print "nombre d'occurrences remplacees : $nb\n";
```

7.3 Translation

Syntaxe	`tr/expression/remplacement/`

Le script ci-dessous lit les caractères minuscules et les transforment en majuscules :

```
#!/usr/bin/perl
print "entrez une phrase : ";
$phrase = <STDIN>;
$phrase =~ tr/[a-z]/[A-Z]/;
print "\n$phrase\n";
```

8 Programmation objet

Vocabulaire :

objet	référence
méthode	procédure
classe	package (stocké dans un fichier .pm)

Méthodes :

new	constructeur
destroy	destructeur appelé automatiquement