

Le langage Perl

Sovanna Tan

Octobre 2007, révision septembre 2008

disponible sur

<http://lacl.univ-paris12.fr/tan/Reseau/perl.pdf>

Plan

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Sources

Documentation de référence en français

- <http://perl.enstimac.fr/>

Cours

- <http://lea-linux.org/cached/index/Dev-perl.html>
- <http://fr.selfhtml.org/cgiperl/langage/index.htm>
- <http://www.mongueurs.net/ressources/perl.html>
- <http://www.perlfr.org/cours/perl.html>
- <http://www.dil.univ-mrs.fr/~chris/Perl/Generalites.htm>
- <http://www.april.org/groupes/doc/perl/perl.html>

Le manuel en ligne

Le manuel est disponible au format NROFF et est accessible via la commande `man`

Les rubriques principales :

<code>perlsyn</code>	Syntaxe de Perl
<code>perldata</code>	Types de données de Perl
<code>perlop</code>	Opérateurs Perl et priorité
<code>perlsub</code>	Les sous-programmes (ou subroutines) de Perl
<code>perlfunc</code>	Fonctions Perl prédéfinies
<code>perlomentut</code>	Apprenez à ouvrir (des fichiers) à la mode Perl
<code>perlpod</code>	Le format Pod (plain old documentation)
<code>perlrun</code>	Comment utiliser l'interpréteur Perl
<code>perldiag</code>	Les différents messages de Perl
<code>perldebug</code>	Débogage de Perl
<code>perlvar</code>	Variables prédéfinies en Perl
<code>perlre</code>	Les expressions rationnelles en Perl
<code>perlref</code>	Références et structures de données imbriquées en Perl
<code>perlform</code>	Formats Perl
<code>perlobj</code>	Objets en Perl
<code>perlnumber</code>	Sémantique des nombres et opérations numériques en Perl
<code>perlsec</code>	Sécurité de Perl
<code>perlmod</code>	Modules Perl (paquetages et tables de symboles)
<code>perlmodlib</code>	Pour construire de nouveaux modules et trouver les existants
<code>perlmodinstall</code>	Installation des modules CPAN
<code>perlutil</code>	Utilitaires livrés avec la distribution de Perl

Ressources

- <http://www.cpan.org/> Comprehensive Perl Archive Network : les logiciels et les bibliothèques
- <http://www.perlfr.org/> Perl en Français dans le texte
- <http://perl.enstimac.fr/> Traduction de la documentation de Perl en français
- Livres O'Reilly
 - *Programming Perl, Third Edition*, Larry Wall, Tom Christiansen, Jon Orwant, 2000

- 1 Ressources
- 2 Introduction**
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Présentation du langage Perl

Practical Extraction and Report Language :

- Créé en 1987 par Larry Wall
- Interprété et pré-compilé à l'exécution
- Licence GNU GPL et Artistic License.

Utilisation

- Administration système
- Manipulation de fichiers textes (XML, courriels, logs, linguistique, génétique)
- Flux et protocoles réseaux (SNMP, web, LDAP, SMTP, ...)
- Bases de données
- Interfaces graphiques (Perl/tk)

Un langage très puissant

- Programmers impérative, fonctionnelle et orientée objet,
- Récursivité, modularité, exceptions,
- Tableaux, listes et tables de hachage (tableaux associatifs) natifs,
- Gestion mémoire : ramasse-miettes,
- Expressions régulières,
- Richesse des bibliothèques (efficacité de programmation),
- Multi plateforme (87 portages),
- Apprentissage facilité (C, sh, sed, POSIX, ...),
- Débugueur intégré.

Inconvénients

- Syntaxe un peu particulière bien que proche du C
- Scripts durs à lire

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage**
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Règles principales

- Définir l'interpréteur en début de script

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

- Les lignes de code se terminent par ;
- Les commentaires sont introduits par le symbole '#'
- Un *bloc* : suite d'instructions, séparées par des points virgules, entourées d'accolades

```
{  
    $toto = 12;  
    print $toto; # 12  
}
```

Variables

- Une variable scalaire (nombre, chaîne de caractères...) est identifié par le symbole \$
ex : \$chaine
- Un tableau est identifié par le symbole @
ex : @tableau,
\$tableau[12] représente l'élément n°12
- Un tableau associatif ou table de hachage est une structure indexée par des chaînes de caractères. Il est identifié par le symbole %

```
%personne=(prenom => 'Sovanna',nom => 'Tan');  
print $personne{'prenom'},' ', $personne{'nom'};  
# Sovanna Tan
```

Déclaration des variables

- La déclaration des variables n'est pas obligatoire.
- La directive `use strict` ; rend les déclarations obligatoires.
Son utilisation est fortement recommandée.

Déclaration des variables globales

Variable globale :

```
use vars qw DELIMa LISTE_VARIABLES_SEP_BLANC DELIM ;
```

^aLes délimiteurs qui ne marchent pas par deux utilisent le même caractère au début et à la fin par contre les quatre sortes de parenthèses (parenthèses, crochets, accolades et inférieur/supérieur) marchent par deux et peuvent être imbriquées.

```
use vars qw($var1 $var2 @tab1 %hash1);
```

```
local ;
```

Changer localement dans un bloc la valeur d'une variable globale :

```
local VARIABLE ;  
local(LISTE_VARIABLE_SEP_VIRGULE);
```

Déclaration des variables locales

Variable locale déclarée avec `my` :

- `my VARIABLE ;`
- `my (LISTE_VARIABLES_SEP_VIRGULE) ;`

```
my $i;  
my($var1, $var2, @tab1, %hash1);
```

Constantes, chaînes de caractères

- Nombres : 123, -123, 0xABCD12, 0123, 1.6e-19
- Chaînes non interprétées : entre les symboles ' ou avec l'opérateur qq DELIM CHAINE DELIM
- Chaînes interprétées : entre les symboles " ou avec l'opérateur qq DELIM CHAINE DELIM

```
$a = "mot";  
print $a;                # mot  
$b = 'Ceci est $a';  
print $b;                # Ceci est $a  
$c = qq(Ceci est aussi $a);  
print $c;                # Ceci est aussi mot
```

Longues chaînes de caractères

Opérateur `<<BALISE BALISE :`

```
print <<ENTETE;
```

```
Content-type: text/html
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html><head><title>Sortie du test</title>
```

```
</head><body>
```

```
ENTETE
```

```
#
```

```
# Content-type: text/html
```

```
#
```

```
# <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
# <html><head><title>Sortie du test</title>
```

```
# </head><body>
```

Les caractères spéciaux

<code>\n</code>	nouvelle ligne (LF,NL)
<code>\r</code>	retour en début de ligne (CR)
<code>\t</code>	tabulation (HT,TAB)
<code>\f</code>	aller à la page suivante (FF)
<code>\a</code>	sonnerie (BEL)
<code>\e</code>	échappement (ESC)
<code>\007</code>	spécifie une valeur octale (007 = sonnerie)
<code>\X7f</code>	spécifie une valeur hexadécimale (7f = efface)
<code>\cZ</code>	caractère de contrôle (CTRL Z)
<code>\\</code>	anti-slash
<code>\"</code>	double quote
<code>\l</code>	force la lettre suivante en minuscule
<code>\L</code>	force en minuscule jusqu'à l'occurrence d'un <code>\E</code>
<code>\u</code>	force la lettre suivante en majuscule
<code>\U</code>	force en majuscule jusqu'à l'occurrence d'un <code>\E</code>
<code>\E</code>	fin de <code>\U</code> ou de <code>\L</code>

Variables prédéfinies

- `man perlvar` donne toutes les variables prédéfinies
- `use English ;` permet d'utiliser les noms des variables au lieu des noms standards des variables spéciales (ex : `$ARG` au lieu de `$_`).

<code>@_ ou @ARG</code>	tableau d'arguments
<code>\$_ ou \$ARG</code>	argument des fonctions unaires
<code>\$0 ou \$PROGRAM_NAME</code>	fichier qui contient le script Perl
<code>\$^O ou \$OSNAME</code>	système d'exploitation
<code>%ENV, \$ENV{"nom_var_environ"}</code>	variables d'environnement
<code>\$! ou \$ERRNO ou \$OS_ERROR</code>	numéro d'erreur <code>errno</code>
<code>\$@ ou \$EVAL_ERROR</code>	message d'erreur du dernier <code>eval</code>
<code>@INC</code>	liste de répertoires contenant les bibliothèques
<code>\$/</code>	séparateur d'enregistrement en lecture, par défaut retour-ligne

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs**
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Opérateurs unaires

- ! négation logique
- moins
- ~ négation bit à bit (complément à 1)
- ++ pré/post incrémentation
- pré/post décrémentation

Opérateurs binaires

<code>+ - * / % **</code>	opérations arithmétiques
<code>.</code>	concatène deux chaînes : "aze"."rt" vaut "azert"
<code>x</code>	"multiplie" une chaîne : "to" x 2 vaut "toto"
<code>& ^</code>	opérations bit à bit
<code><< >></code>	décalage
<code>&& and</code>	et logique
<code> or</code>	ou logique

Affectation

<code>=</code>	affectation standard
<code>+= -= *= /= %=</code>	pour les opérations arithmétiques
<code>&&= = !=</code>	pour les opérations logiques
<code>&= = ^= ~=</code>	pour les opérations logiques sur les bits
<code><<= >>=</code>	pour les décalages
<code>.=</code>	concaténation de chaînes de caractères

Opérateurs de comparaison

< <= > >=	comparaison de nombres
== !=	égal, différent pour les nombres
lt le gt ge	comparaison de chaînes de caractères
eq ne	égal, différent pour les chaînes de caractères
=~ !~	correspond, ne correspond pas à une expr. rég.

Les opérateurs de comparaison $\leq$ $\geq$ (spaceship) et cmp

Pour les nombres :

Relation entre \$a et \$b	Opération	Valeur de \$x
<code>\$a > \$b</code>	<code>\$x = \$a <=> \$b ;</code>	+1
<code>\$a = \$b</code>	<code>\$x = \$a <=> \$b ;</code>	0
<code>\$a < \$b</code>	<code>\$x = \$a <=> \$b ;</code>	-1

Pour les chaînes de caractères :

Relation entre \$a et \$b	Opération	Valeur de \$x
<code>\$a gt \$b</code>	<code>\$x = \$a cmp \$b ;</code>	+1
<code>\$a eq \$b</code>	<code>\$x = \$a cmp \$b ;</code>	0
<code>\$a lt \$b</code>	<code>\$x = \$a cmp \$b ;</code>	-1

Opérateurs pour les expressions régulières

Opérateurs de construction :

<code>m/MOTIF/cgimosx</code> ou <code>/MOTIF/cgimosx</code>	recherche (matche)
<code>?MOTIF?</code>	recherche de la première occurrence
<code>q/CHAINE/</code> ou <code>'CHAINE'</code>	chaîne non interprétée
<code>qq/CHAINE/</code> ou <code>"CHAINE"</code>	chaîne interprétée
<code>qr/CHAINE/imosx</code>	traite CHAINE comme une expr. rég.
<code>qx/CHAINE/</code> ou <code>'CHAINE'</code>	appel système
<code>qw/CHAINE/</code>	renvoie la liste des mots de CHAINE
<code>s/MOTIF/REMPLACEMENT/egimosx</code>	substitution
<code>tr/AREMPLACER/REMPLACES/cds</code>	remplacement de caractères
ou <code>tr/ATRADUIRE/TRADUITS/cds</code>	

Opérateurs d'application :

<code>=~</code>	correspond (matche)
<code>!~</code>	ne correspond pas

Opérateurs sur les fichiers

- d répertoire
- e fichier existant
- s taille du fichier
- z fichier de taille nulle
- M âge du fichier en jours à partir de la date d'exécution du script

Certaines opérations sont spécifiques à Unix :

- r fichier accessible en lecture par la personne qui exécute le script
- w fichier accessible en écriture par la personne qui exécute le script
- x fichier exécutable
- o fichier possédé par la personne qui exécute le script
- g bit setgid posé (exécution avec gid du fichier ou création avec gid du répertoire)
- k bit sticky posé (seuls root et propriétaire peuvent effacer)
- l lien symbolique

Appel système

Opérateur ' ou qx/ / :

```
$date='date';  
$hostname='hostname';  
print $date,' ', $hostname;  
# Fri Sep 19 11:22:16 CEST 2008  
# info.univ-paris12.fr
```

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle**
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Vrai/Faux

Faux :

- Le nombre 0
- La chaîne de caractères '0'
- La chaîne de caractère ''
- La liste vide ()
- La valeur undef

Vrai :

Tout autre valeur est considérée comme vraie.

La négation d'une valeur vraie obtenue avec ! ou not vaut 0 si elle est évaluée comme un nombre, '' si elle est évaluée comme une chaîne de caractères.

Les conditions if, ? :, unless

if :

- INSTRUCTION if (CONDITION)
- if (CONDITION) BLOC
- if (CONDITION) BLOC else BLOC
- if (CONDITION) BLOC elsif (COND) BLOC ... else BLOC

Les accolades sont obligatoires.

Opérateur conditionnel comme en C :

(CONDITION) ? EXPR : EXPR

unless :

- INSTRUCTION unless (CONDITION)
- unless (CONDITION) BLOC

Exemple avec unless

Saisie d'une chaîne de caractère non vide sur l'entrée standard :

```
#!/usr/bin/perl
use strict;
my $a;
print "Entrer un entier\n";
$a=<STDIN>;
unless($a<0){
    print("Cet entier est positif\n");
}
```

Entrer un entier

6

Cet entier est positif

Les boucles while, until

while :

- [LABEL] while (CONDITION) BLOC
- [LABEL] while (CONDITION) BLOC continue BLOC

Le bloc continue est équivalent à la troisième partie de la boucle for du C.

until :

- [LABEL] until (CONDITION) BLOC
- [LABEL] until (CONDITION) BLOC continue BLOC

Exemple avec until

Saisie d'une chaîne de caractère non vide sur l'entrée standard :

```
#!/usr/bin/perl
use strict;
use vars qw($name);
until(length($name)){
    print("What is your name? ");
    $name = <STDIN>;
    chomp($name);
    print("Msg: Zero length input. Please try again\n")
        if(! length($name));
}
print 'Your name is ', $name, "\n";
```

```
What is your name?
Msg: Zero length input. Please try again
What is your name? Tan
Your name is Tan
```

La boucle for similaire au C

`for :`

`[LABEL] for (INITIALISATION ; CONDITION ; EXPR) BLOC`

Si des variables sont déclarées par `my` dans la section d'initialisation d'un `for`, la portée lexicale de ces variables est limitée au corps de la boucle et sa section de contrôle.

La boucle foreach, for

foreach ou for :

- LABEL foreach VAR (ARRAY) BLOC
- LABEL foreach VAR (ARRAY) BLOC continue BLOC

On peut employer le mot clé for à la place de foreach.

Exemple :

```
foreach $fichier ('ls'){      # ex1.pl
    print $fichier             # ex2.pl
}                              # ex3.pl

%capitales = ('Chine' => 'Pekin', 'Angleterre' => 'Londres',
              'France' => 'Paris', 'Italie' => 'Rome');
foreach $k (keys (%capitales)) {
    print $k, ", ", $capitales{$k}, '|';
# France,Paris|Angleterre,Londres|Italie,Rome|Chine,Pekin|
}
```

L'instruction de contrôle next

`next` (similaire à `continue` en C) :

- `next LABEL` : démarre la prochaine itération de la boucle identifiée par `LABEL`
- `next` : lorsque `LABEL` est omis, la commande se réfère au bloc englobant le plus intérieur

S'il y a un bloc `continue`, il est exécuté.

L'instructions de contrôle last

last (similaire à break en C) :

- last LABEL : permet de sortir immédiatement de la boucle identifiée par LABEL
- last : lorsque LABEL est omis, la commande se réfère à la boucle englobante la plus profonde

S'il y a un bloc continue, il n'est pas exécuté.

Exemple avec next et last

Recherche des couple d'entiers dont le produit vaut 36 :

```
#!/usr/bin/perl;
my $v = 36;
EXTERNE: for ($i = 1; $i <= 10; $i++) {
    INTERNE: for ($j = 1; $j <= 10; $j++) {
        print "$i $j|";
        if ($i * $j == $v) {
            print "$v = $i * $j\n";
            next EXTERNE;
        }
        last INTERNE if ($j >= $i);
    }
}
```

```
# 1 1|2 1|2 2|3 1|3 2|3 3|4 1|4 2|4 3|4 4|5 1|5 2|5 3
# 5 4|5 5|6 1|6 2|6 3|6 4|6 5|6 6|36 = 6 * 6
# 7 1|7 2|7 3|7 4|7 5|7 6|7 7|8 1|8 2|8 3|8 4|8 5|8 6
# |8 7|8 8|9 1|9 2|9 3|9 4|36 = 9 * 4
```

L'instructions de contrôle redo

redo :

- redo LABEL : répétition de l'itération de la boucle identifiée par LABEL
- redo : lorsque LABEL est omis, la commande se réfère à la boucle englobante la plus profonde

S'il y a un bloc continue, il est exécuté.

Exemple avec redo

Saisie d'une chaîne de caractère non vide sur l'entrée standard :

```
#!/usr/bin/perl
use strict;
use vars qw($name);
print 'What is your name ? ';
LINE: {
    chomp($name=<STDIN>);
    if(! length($name)) {
        print("Msg: Zero length input. Please try again\n");
        redo LINE;
    }
}
print 'Your name is ', $name, "\n";
# What is your name?
# Msg: Zero length input. Please try again
# What is your name? Tan
# Your name is Tan
```

Récapitulation next, last, redo

```
while (EXPR) {  
    ### redo vient toujours ici  
    do_something;  
} continue {  
    ### next vient toujours ici  
    do_something_else;  
    # puis retour au sommet pour revérifier EXPR  
}  
### last vient toujours ici
```

L'instruction eval

`eval` `EXPR`; L'expression est parcourue puis exécutée.
`eval` `BLOC` Le bloc est parcouru une seule fois à la compilation et exécuté.

`eval` `BLOC` n'est pas considéré comme une boucle, on ne peut pas utiliser les instructions de contrôle de boucle `next`, `last` et `redo`.

```
$z=eval '$OSNAME'; # équivalent à $z=$OSNAME
print $z, "\n";
#
# linux
```

L'instruction `do`

`do BLOC`

- Renvoie la valeur de la dernière commande.
- Lorsqu'il est suivi de `while` ou `until`, exécute le bloc une fois avant la boucle.
- N'est pas considéré comme une boucle, on ne peut pas utiliser les instructions de contrôle de boucle.

`do FICHER;    similaire à eval `cat FICHER ``

- Plus efficace
- Une trace du fichier courant est gardée pour les messages d'erreur.

L'instruction die

die LISTE_SEP_VIRGULE

- En dehors d'un bloc `eval`, affiche la liste sur la sortie d'erreur `STDERR` et quitte le programme
- Dans un bloc `eval`, le message d'erreur est mis dans `$@` et `eval` s'achève sur la valeur `undef`.

```
chdir '/usr/spool/news' or die "Can't cd to spool: $!\n"  
#  
# Can't cd to spool: No such file or directory
```

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions**
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Les références

Une référence se construit avec l'opérateur `\` :

- `$refx=\$x`
- `$refy=\@y`
- `$refz= \%z`

Valeur pointée par une référence :

- `${$refx}` ou `$$refx`
- `@{$refy}` ou `@$refy` et pour les éléments `$refy->[2]` ou `@{$refy}[2]` ou `@$refy[2]`
- `%{$refz}` ou `$$refz` et pour les éléments `$refz->{nom}` ou `${$refz}{nom}` ou `$$refz{nom}`

Les sous-programmes ou fonctions

Définition :

```
sub NAME BLOCK
```

Déclaration :

```
sub NAME ;
```

- Une fonction peut être définie n'importe où.
- Avec `use strict ;`, il faut déclarer les fonctions au début du script.
- Dans la définition des fonctions, déclaration des variables locales avec `my`.

Passage des paramètres

- Les paramètres d'un sous-programme constituent un tableau noté `@_`. Ses éléments sont `$_[0]`, `$_[1]`, ..., `$_[ $#_ ]`.
- Les paramètres sont passés par valeur.
- On peut passer des paramètres par référence pour modifier leurs valeurs.

Valeur de retour

- Un sous-programme renvoie une valeur ou un tableau de valeurs.
- S'il n'y a pas de `return` la valeur de la dernière expression est renvoyée.
- On peut renvoyer des références pour des structures plus complexes.

Appel d'un sous-programme

<code>&NAME(LIST) ;</code>	
<code>NAME(LIST) ;</code>	le <code>&</code> est optionnel avec les parenthèses
<code>NAME LIST ;</code>	si le sous-programme est prédéclaré ou importé
<code>&NAME ;</code>	équivalent à <code>NAME(@_)</code>
<code>NAME ;</code>	équivalent à <code>NAME()</code>

Exemple :

```
sub max {  
    my $max = $_[0];  
    foreach $foo (@_) {  
        $max = $foo if $max < $foo;  
    }  
    return $max;  
}  
$z=max 1,3,7,8,2,3;  
print $z, "\n"; # 8
```

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules**
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles

Paquetage

package :

package NOM définition d'un espace de nommage

- Visibilité jusqu'à la fin du bloc ou la fin de fichier
- Une table des symboles %paquetage:: est associée à paquetage.

Accès aux éléments de l'extérieur :

\$paquetage : :variable

@paquetage : :tableau

%paquetage : :hachage

&paquetage : :fonction

Module

Constructeurs et Destructeurs de paquetage :

BEGIN { } bloc exécuté dès que possible
END { } bloc exécuté aussi tard que possible

Module :

Un module est un paquetage défini dans un fichier de même nom et l'extension `.pm` destiné à être réutilisé.

Utiliser d'autres fichiers

require :

```
require Foo::Bar;      cherche Foo/Bar.pm dans @INC
require 'fichier.pl';  cherche fichier.pl dans @INC
```

use :

```
use Module;            équivalent à BEGIN { require Module;
                        import Module; }
use Module LIST;        équivalent à BEGIN { require Module;
                        import Module LIST; }
                        Certains éléments viennent dans l'espace de
                        nom courant.
use MODULE ()           équivalent à BEGIN { require Module; }
                        L'espace de nom courant n'est pas modifié.
```

La fonction `import` doit être définie dans le module. Elle peut être héritée du module `Exporter`.

La programmation objet en Perl

Classe :

Une classe est un module qui fournit les sous-programmes qui peuvent être utilisés comme méthodes.

Méthode :

Une méthode est juste un sous-programme qui prévoit que son premier argument est le nom d'un module ou une référence.

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières**
- 9 Entrées/Sorties
- 10 Fonctions utiles

Expression régulière

Une *expression régulière* ou *rationnelle* est un masque sous forme de suite de caractères qui permet de décrire le contenu d'une chaîne de caractères, afin de tester si cette dernière correspond à un motif, d'en extraire des informations ou bien d'y effectuer des substitutions.

- Syntaxe héritée d'Unix, semblable à la base à celle de la commande `grep`
- Fonctionnalités supplémentaires
- Pratique pour analyser des fichiers texte
- Pratique pour contrôler les entrées des scripts CGI
- Tutoriels accessibles avec `man`, `perlrequick` et `perlretut`

Les méta-caractères :

<code>\</code>	annule le meta-sens du meta-caractère qui suit
<code>^</code>	début de ligne
<code>.</code>	n'importe quel caractère (sauf <code>\n</code>)
<code>\$</code>	fin de ligne
<code> </code>	alternative
<code>()</code>	groupement et capture <code>\$1</code> , <code>\$2</code> ,... correspondent aux expressions entre parenthèses
<code>[]</code>	classe de caractères
<code>[^]</code>	classe de caractères complément

Les caractères spéciaux

Tous les caractères spéciaux de la diapositive 20 peuvent être utilisés dans les expressions régulières.

Autres caractères spéciaux :

- `\w` caractère de mot (alphanumérique plus "_")
- `\W` caractère autre que de mot
- `\s` un caractère d'espace.
- `\S` caractère autre qu'un espace
- `\d` un chiffre
- `\D` un caractère autre qu'un chiffre
- `\pP` la propriété P (nommée). Utiliser `\p{Prop}` pour les noms longs
- `\X` une séquence d'un caractère unicode étendue
- `\C` un caractère d'un seul octet

Les classes de caractères

[:CLASSE_POSIX:]	\p{PROPRIÉTÉ_UNICODE}	backslash
alpha	IsAlpha	
alnum	IsAlnum	
ascii	IsASCII	
blank	IsSpace	
cntrl	IsCntrl	
digit	IsDigit	\d
graph	IsGraph	
lower	IsLower	
print	IsPrint	
punct	IsPunct	
space	IsSpace ou IsSpacePerl	\s
upper	IsUpper	
word	IsWord	\w
xdigit	IsXDigit	

Les assertions de longueur nulle

- `\b` limite d'un mot
- `\B` autre chose qu'une limite de mot
- `\A` début de la chaîne
- `\Z` fin de chaîne ou juste avant le saut de ligne
- `\z` fin de chaîne
- `\G` endroit où s'est arrêté le précédent `m/.../g`

Les quantificateurs

- Les quantificateurs gourmands matchent la plus grande chaîne possible.
- C'est le comportement classique des expressions régulières

Les quantificateurs gourmands :

*	reconnaît 0 fois ou plus
+	reconnaît 1 fois ou plus
?	reconnaît 0 ou 1 fois
{n}	reconnaît n fois exactement
{n,}	reconnaît au moins n fois
{n,m}	reconnaît au moins n fois mais pas plus de m fois

En Perl, on dispose de quantificateurs qui matchent la plus petite chaîne possible.

Les quantificateurs non gourmands :

*?	reconnaît 0 fois ou plus
+?	reconnaît 1 fois ou plus
??	reconnaît 0 ou 1 fois
{n}?	reconnaît n fois exactement
{n,}?	reconnaît au moins n fois
{n,m}?	reconnaît au moins n fois mais pas plus de m fois

Exemple gouton/non glouton

```
$chaine = 'Voila un <a href="index.html">index</A>';  
$chaine =~ /(<.+>)/;  
$greedy=$1;  
$chaine =~ /(<.+?>)/;  
$nongreedy=$1;  
print "chaine : ", $chaine, "\n";  
print "contient <.+> : ", $greedy, "\n";  
print "contient <.+?> : ", $nongreedy, "\n";
```

affiche

```
chaine : Voila un <a href="index.html">index</A>  
contient <.+> : <a href="index.html">index</A>  
contient <.+?> : <a href="index.html">
```

La reconnaissance de motif ou pattern matching

L'opérateur `m/MOTIF/cgimosx` ou `/MOTIF/cgimosx` :

- Lorsque `/` est utilisé comme délimiteur, `m` est optionnel.
- Lorsque `m` est utilisé, on peut utiliser n'importe quelle paire de caractères ni alphanumériques ni blancs comme délimiteurs.

Les modificateurs :

- `c` ne pas réinitialiser la position de recherche lors d'un échec avec `/g`
- `g` recherche globale, trouver toutes les occurrences
- `i` reconnaissance de motif indépendamment de la casse (maj/min)
- `m` traitement de chaîne comme étant multi-lignes
- `o` compilation du motif uniquement la première fois
- `s` traitement de la chaîne comme étant une seule ligne
- `x` utilisation des expressions rationnelles étendues

Exemple de recherche globale

```
$chaine = 'Un <a href="index.html">index</a><p>Fin</p>';  
@liste= ($chaine =~ /<.+?>/g);  
foreach $i (@liste){  
    print $i, "\n";  
}
```

affiche

```
<a href="index.html">  
</a>  
<p>  
</p>
```

La substitution

L'opérateur `s/MOTIF/REPLACEMENT/egimosx` :

- recherche le motif dans une chaîne
- s'il est trouvé, le substitue par le texte `REPLACEMENT`
- renvoie le nombre de substitutions effectuées ou la chaîne vide

Les options :

- e évaluer la partie droite comme une expression
- g substitution globale pour toutes les occurrences
- i motif indépendant de la casse (majuscules/minuscules)
- m traitement de chaîne comme étant multi-lignes
- o compilation du motif uniquement la première fois
- s traitement de la chaîne comme étant une seule ligne
- x utilisation des expressions rationnelles étendues

```
$chaine="J\'ai pris 2x2 feuilles.";
print "chaine : ", $chaine, "\n";
$chaine=~s/x/fois/g;
print "chaine : ", $chaine, "\n";
$chaine=~s/2/ deux /g;
print "chaine : ", $chaine, "\n";
```

affiche

```
chaine : J\'ai pris 2x2 feuilles.
chaine : J\'ai pris 2fois2 feuilles.
chaine : J\'ai pris  deux fois deux  feuilles.
```

La transcription

L'opérateur `tr/ATRADUIRE/TRADUITS/cds` ou
`y/ATRADUIRE/TRADUITS/cds` :

- remplace chaque caractère figurant dans ATRADUIRE par le caractère figurant à la même place dans TRADUITS
- un intervalle de caractères peut être utilisé
- ne reconnaît pas les classes de caractères comme la commande unix `tr`

Les options :

- `c` complémente la liste ATRADUIRE
- `d` efface les caractères trouvés mais non remplacés
- `s` agrège les caractères de remplacement dupliqués

Exemple de transcription

```
$chaine="Il fait beau.";
print "chaine : ", $chaine, "\n";
$chaine=~tr/a-z/A-Z/;
print "chaine : ", $chaine, "\n";
$chaine= <<FIN;
Ce dernier point
est      très      pratique pour
pour     comptabiliser les occurrences
FIN
print "chaine : ", $chaine;
$chaine=~tr/a-zA-Z/ /cs;
print "chaine : ", $chaine, "\n";
```

Le deuxième tr remplace tout caractère qui n'est pas une lettre par un espace.

```
chaine : Il fait beau.
chaine : IL FAIT BEAU.
chaine : Ce point
est      très      pratique pour
pour     compter les occurrences
chaine : Ce point est tr s pratique pour pour compter les occurrences
```

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties**
- 10 Fonctions utiles

Manipulation de fichiers

Pour manipuler un fichier, on lui associe un descripteur à l'ouverture.

Les descripteurs de fichiers spéciaux :

STDIN	entrée standard
STDOUT	sortie standard
STDERR	sortie d'erreur

Fermeture d'un fichier :

close	DESCRIPTEUR	fermeture du fichier
-------	-------------	----------------------

Ouverture d'un fichier

La fonction open :

<code>open(DESCRIPTEUR,MODE,EXPR)</code>	ouvre le fichier dont le nom est donné par EXPR et l'associe à DESCRIPTEUR
<code>open(DESCRIPTEUR,EXPR)</code>	le mode par défaut est lecture

Les modes d'ouverture :

<	lecture
>	écriture
>>	ajout
+<	mise à jour en lecture/écriture
+>	écrasement préalable et mise à jour en lecture/écriture

Exemple :

```
open(FILE,'>',"fichier") or die "Cannot open fichier: $!";
```

Les fonctions print et printf

La fonction print :

<code>print DESCRIPTEUR LISTE</code>	écrit une chaîne de caractères ou une liste de chaînes dans DESCRIPTEUR
<code>print LISTE</code>	affiche sur la sortie standard

La fonction printf :

<code>printf DESCRIPTEUR FORMAT, LISTE</code>	similaire à <code>fprintf</code> du C
<code>printf FORMAT, LISTE</code>	similaire à <code>printf</code> du C

L'usage de `print` est recommandé s'il est suffisant.

Exemple :

```
$number=10/3;  
printf("%.3f\n", $number); # 3.333
```

L'opérateur < > (diamond)

<DESCRIPTEUR> renvoie la ligne suivante du fichier

- La ligne est lue jusqu'à la fin de ligne définie par \$/.
- A la fin du fichier <DESCRIPTEUR> renvoie faux.

Exemple :

```
open(FILE, "fichier") or die "Cannot open fichier: $!";  
$i=0;  
while(<FILE>){  
  continue{  
    $i++;  
  }  
  print "Le fichier a $i lignes\n";  
  close(FILE);  
}
```

- 1 Ressources
- 2 Introduction
- 3 Éléments du langage
- 4 Opérateurs
- 5 Structures de contrôle
- 6 Fonctions
- 7 Modules
- 8 Expressions régulières
- 9 Entrées/Sorties
- 10 Fonctions utiles**

Fonctions numériques, fonctions sur les tableaux

Fonctions numériques :

abs	atan2	cos	exp	hex	int
log	oct	rand	sin	sqrt	srand

Fonctions sur les tableaux :

pop TAB	dépile et renvoie la dernière valeur du tableau
push TAB, LISTE	empile les valeurs de la liste à la fin du tableau
shift TAB	renvoie et supprime la première valeur du tableau
splice TAB,OFFSET,LG,LISTE	supprime LG éléments du tableau à partir de OFFSET et les remplace par les éléments de LISTE
splice TAB,OFFSET,LG	supprime LG éléments à partir de OFFSET
splice TAB,OFFSET	supprime tout à partir de OFFSET
splice TAB	supprime tout
unshift TAB, LISTE	ajoute les valeurs de LISTE au début du tableau, retourne le nouveau nombre d'éléments

fonctions sur les hachages, fonctions sur les listes

Fonctions sur les hachages :

<code>delete EXPR;</code>	efface un élément ou une partie du hachage
<code>each HASH;</code>	renvoie la prochaine liste (clé,valeur) ou prochaine clé
<code>exists EXPR;</code>	renvoie vrai si l'élément existe
<code>keys HASH;</code>	renvoie la liste des clés
<code>values HASH;</code>	renvoie la liste des valeurs

Fonctions sur les listes :

<code>grep BLOC LISTE</code> ou <code>grep(EXPR, LISTE)</code>	similaire au <code>grep</code> unix
<code>join(EXPR,LISTE)</code>	concatène les chaînes de LISTE en les séparant par EXPR
<code>map BLOC LISTE</code> ou <code>map(EXPR,LISTE)</code>	évalue le bloc BLOC ou l'expression EXPR pour chaque élément de LISTE, renvoie la liste de tous les résultats
<code>reverse LISTE</code>	renvoie la liste des éléments en ordre inverse
<code>sort SUBNAME LISTE</code> ou <code>sort BLOC LISTE</code> ou <code>sort LISTE</code>	renvoie la liste des valeurs triées avec SUBNAME tri avec un sous-programme anonyme tri dans l'ordre de comparaison des chaînes de caractères

Fonctions sur les scalaires et les chaînes de caractères

<code>chomp VAR</code>	supprime toute fin de ligne correspondant à la valeur courante de \$/
<code>ou chomp(LISTE)</code>	s'applique à \$_
<code>ou chomp</code>	efface et retourne le dernier caractère d'une chaîne
<code>chop VAR</code>	
<code>chop(LISTE) ou chop</code>	renvoie la position de la première occurrence de SUBSTR dans CHAINE à partir de POS inclus
<code>index(CHAINE,SUBSTR,POS)</code>	renvoie la longueur en caractères de valeur de EXPR
<code>index(CHAINE,SUBSTR)</code>	renvoie la valeur de EXPR avec tous les caractères non alphanumériques précédés par un \
<code>length EXPR</code>	renvoie la position de la dernière occurrence de SUBSTR dans CHAINE à partir de POS inclus
<code>quotemeta EXPR</code>	découpe la chaîne EXPR en une liste de chaînes et retourne la liste
<code>rindex(CHAINE,SUBSTR,POS)</code>	extraît une sous-chaîne de EXPR et la renvoie
<code>rindex(CHAINE,SUBSTR)</code>	
<code>split(/MOTIF/,EXPR,LIM)</code>	
<code>split(/MOTIF/,EXPR)</code>	
<code>substr(EXPR,OFFSET,LONG)</code>	

Fonctions sur les fichiers

Fonctions d'entrée/sortie :

binmode	close	closedir	die	eof	fileno
flock	format	getc	print	printf	read
readdir	rewinddir	seek	seekdir	select	syscall
sysread	sysseek	syswrite	tell telldir	truncate	
warn	write				

Fonctions pour données de longueur fixe ou pour enregistrements :

pack	read	syscall	sysread
syswrite	unpack	vec	

Fonctions de descripteurs de fichiers, de fichiers ou de répertoires :

chdir	chmod	chown	chroot	fcntl	glob
ioctl	link	lstat	mkdir	open	opendir
readlink	rename	rmdir	stat	symlink	sysopen
umask	unlink	utime			

Fonctions sur les processus, les utilisateurs et les groupes

Fonctions de gestion des processus et des groupes de processus :

alarm	exec	fork	getpgrp	getppid	getpriority
kill	pipe	setpgrp	setpriority	sleep	system
times	wait	waitpid			

Manipulation des informations sur les utilisateurs et les groupes :

endgrent	endhostent	endnetent	endpwent	getgrent	getgrgid
getgrnam	getlogin	getpwent	getpwnam	getpwuid	setgrent
setpwent					

Fonctions IPC (communication inter-processus) System V :

msgctl	msgget	msgrcv	msgsnd	semctl	semget
semop	shmctl	shmget	shmread	shmwrite	

Fonctions de date et heure, fonctions liées au réseau

Fonction de date et heure :

gmtime localtime time times

Fonctions de bas niveau liées aux sockets :

accept	bind	connect	getpeername	getsockname	getsockopt
listen	recv	send	setsockopt	shutdown	socket
socketpair					

Manipulation des informations du réseau :

endprotoent	endservent	gethostbyaddr	gethostbyname
gethostent	getnetbyaddr	getnetbyname	getnetent
getprotobyname	getprotobynumber	getprotoent	getservbyname
getservbyport	getservent	sethostent	setnetent
setprotoent	setservent		

Merci.