Programmer en Perl

François Gannaz – SILECS

Formation Continue Grenoble INP

Plan

Au commencement...

- Larry Wall, le fondateur
  - diplômé de *linguistique*
  - programmeur émérite en C
  - créateur de *patch*
  - Trois vertus du programmeur : *Laziness, Impatience and Hubris*
- Perl 1 en 1987, Perl 4 stable en 91
  Sous licence artistique (et GPL)
- **Perl 5** en 1994
  Récriture complète du langage Perl.
- Le futur : Perl 6, débuté en 2002
  Nouveau langage, différent de Perl 5.
  Implémenté par *Rakudo star* sur la machine virtuelle *Parrot*.

Perl, c'est-à-dire...

Le nom de Perl a pour origine :

- Une citation des évangiles,
  "A pearl of great price", Matthew, 13 :46.
- Pratical Extraction and Report Language
  Langage pratique d'extraction et de génération de rapports.
- Pathologically Eclectic Rubbish Lister
  Listeur pathologique de débris éclectiques.

Les caractéristiques de Perl

Perl est un langage **interprété**.
Un programme perl n'est pas compilé pour produire un exécutable indépendant.

Spécialités Perl :

- Le traitement du texte
  En particulier, les expressions régulières.
- Structures évoluées
  Langage de haut niveau : des types de données complexes sont intégrés au langage
- Souplesse
  There is more than one way to do it.

Du point de vue du développeur, la tâche est simplifiée :

- Syntaxe proche du C, de sed et de sh
- Faiblement typé (conversions transparentes)

Domaine de compétence

Perl n'est pas recommandé pour :

- Exécutables autonomes
- Performance cruciale en temps ou en mémoire
- Programmation bas niveau (drivers, noyau)

Perl est adapté et utilisé pour :

- Manipulation de texte
- Web
- Administration système
- Domaines spécialisés (bioinformatique, etc.)

Documentation

La documentation officielle de Perl est excellente et regorge d'exemples.

Doc générale

Table des matières : `perldoc perl`

Par ex., fonctions par catégories : `man perlfunc`

Rechercher une fonction

`perldoc -f` *`fonction`*

Rechercher dans la FAQ

`perldoc -q` *`motif`*

En ligne

- `perl.org` Le site officiel du langage.
- `cpan.org` Le dépôt central des modules Perl.

La licence artistique

Perl à la réputation de produire du code *sale* :

- faiblement typé
- *There is more than one way to do it.*

On peut imposer une dose de rigueur à Perl, mais ce ne sera jamais du Java.

> *Les langues sont fondamentalement amorales. La langue n'est pas le niveau auquel on devrait obliger à "penser bien". On ne peut garantir la moralité par la syntaxe.*
>
> Larry Wall

Installation sous Linux

Perl est déjà installé !

Éditeurs

- **Vim** et **Emacs** (cperl-mode) sont excellents.
- Beaucoup d'éditeurs libres légers :
  de **Gedit** (généraliste) à **Geany** (orienté code).
- IDE multi plates-formes :
  - **Eclipse** + EPIC : attention, très lourd
    Pour tous langages, dont Perl.
    `http://www.epic-ide.org/`
  - **Komodo Edit**
    Pour Perl, Python, Ruby, PHP.
    `http://www.activestate.com/komodo-edit/`
  - **Padre** : uniquement pour du code Perl
    En cours de développement
    Debian : `aptitude install padre/unstable`

Installation sous Mac OS X

Perl est déjà installé !

Éditeurs

- Les éditeurs du monde Unix :
  Vim et **Emacs** (Carbon Emacs)
- Beaucoup d'éditeurs payants :
  TextMate, BBedit, Affrus. . .
- IDE multi plates-formes :
  - **Eclipse** + EPIC : attention, très lourd
    Pour tous langages, dont Perl.
    http://www.epic-ide.org/
  - **Komodo Edit**
    Pour Perl, Python, Ruby, PHP.
    http://www.activestate.com/komodo-edit/

Installation sous Windows

Perl n'est **pas installé** !

Deux distributions possibles :

- **ActivePerl**, http://www.activestate.com/activeperl
  support commercial, meilleure intégration à Windows.
- **Strawberry Perl**, http://strawberryperl.com/
  distribution plus ouverte, proche de UNIX.

Éditeurs

- Les classiques sous Windows comme **Notepad++** (libre).
- Les portages venant d'Unix : **gvim** et **Emacs**.
- IDE multi plates-formes :
  - **Eclipse** + EPIC : attention, très lourd
    http://www.epic-ide.org/
  - **Komodo Edit**
    http://www.activestate.com/komodo-edit/
  - **Padre** : uniquement pour du code Perl
    Utiliser de préférence le paquet StrawberryPerl+Padre
    http://padre.perlide.org/download.html

Le premier programme

```
print "Hello, world!\n";
```

hello.pl

Exécution :

```
foo@bar> perl hello.pl
```

Le premier programme

Intégration dans unix

```
#!/usr/bin/perl
print("Hello, world!\n");
```

hello2.pl

Exécution :

```
foo@bar> chmod u+x hello2.pl
foo@bar> ./hello2.pl
```

Intégration dans Windows

Associer l'extension `.pl` avec l'interpréteur Perl (`perl.exe`).

Premiers exemples

One-liner

```
% perl -i.BAK -C -pe 's#<(/?\p{Ll}+)>#<\L$1>#' *.xml
```

Met en minuscule toutes les balises des fichiers XML en créant une copie de sauvegarde en .xml.BAK.

Premiers exemples

Script

```
#!/usr/bin/perl -w
foreach (glob "*") {
    s/\d/*/g;
    s/\.[^\.]+?$//;
    $motif{$_}++;
}
foreach (sort keys %motif) {
    print "'$_' $motif{$_} occurrences.\n";
}
```

Regroupe les fichiers selon leur nom privé de chiffres, et affiche le nombre d'occurrences.

Exemple d'utilisation de Perl

Situation

Répertoire contenant 999 fichiers :

file1.html ...file50.html ...file999.html

Objectif

Les renommer en :

file001.html ...file050.html ...file999.html

Solution

Utiliser la commande unix `rename`.

```
rename 's/^file(\d)\./file0$1./'
rename 's/^file(\d\d)\./file0$1./'
```

Le programme rename

```
#!/usr/bin/perl -w
# rename - Larry's filename fixer
$op = shift or die "Usage: rename expr [files]\n";
chomp(@ARGV = <STDIN>) unless @ARGV;
for (@ARGV) {
    $was = $_;
    eval $op;
    die $@ if $@;
    rename($was,$_) unless $was eq $_;
}
```

C'est un script Perl !

```
#!/usr/bin/perl -w
# rename - Larry's filename fixer
$op = shift or die "Usage: rename expr [files]\n";
chomp(@ARGV = <STDIN>) unless @ARGV;
for (@ARGV) {
    $was = $_;
    eval $op;
    die $@ if $@;
    rename($was,$_) unless $was eq $_;
}
```

Commentaires

- # : commentaires
- {...} : bloc
- ; : fin de commande
- indentation non signifiante

Plan

Variables

Déclaration

La déclaration n'est pas obligatoire.

```
#!/usr/bin/perl
$variable = 13;
```

La variable est créée à la première utilisation.
Par défaut, elle vaut toujours `undef`.

Programmation *propre*

La directive `use warnings` teste la cohérence du code.
Et `use strict` impose une **déclaration préalable** avec `my`.

```
#!/usr/bin/perl -w
use strict;
use warnings;
my $variable;
$variable = 14;
```

Types de données

Perl est *faiblement typé*.

Perl connaît 3 types de données :

- **scalaires**
  Données non-composées : nombres, chaînes de caractères,...

```
1      -10.02      "texte"
```

- **tableaux**
  Tableaux unidimensionnels de **scalaires**

```
(1, "deux", 3, "quatre")
```

- **tableaux associatifs** (tables de hachage, *hashes*)
  À un **scalaire** donné, on associe un **scalaire**.

```
( "un" => 1 , "deux" => 2, -1 => 1 )
```

Scalaires

Ce sont toutes les données simples (*non-composées*) de Perl.

Une variable scalaire s'écrit `$a`, `$var`,...

```
$n = 14;              $hex=0xff ;
$texte = "Perl" ;     my $txt='Perl 5' ;
```

Les scalaires peuvent se ranger en 3 catégories :

- nombre
- texte
- référence

Perl se charge des conversions, elles ne sont pas explicites.

Les scalaires : les nombres

Opérateurs numériques

Identiques à ceux du C.

Affectation : `=`

Calcul : `+` `-` `*` `/`

Combiné : `+=` `-=` `*=` `/=`

Modulo : `%`

Comparaison : `==` `<` `<=` `>` `>=` `!=`

Incrémentation/décrémentation : `++` `--`

Opérateurs supplémentaires.

Exponentiation : `**`

Comparaison : `<=>`

Les scalaires : les chaînes de caractères

Interpolation

- Sans interpolation

```
$t='Salut !';              Salut !
$t='Salut\n a \\toi';      Salut\n␣a␣\toi
$t='j\'arrive';            j'arrive
```

  Tous les caractères sont conservés, sauf `\\` et `\'`.

- Avec interpolation

```
$t="Salut\n a \\toi";      Salut
                           ␣a␣\toi
$a='Perl';
$b="Le chameau de $a";     Le␣chameau␣de␣Perl

$j=40;
$t="Les $j voleurs";       Les␣40␣voleurs
```

Les scalaires : les chaînes de caractères

Opérateurs

Concaténation : `.`

Réplication : `x`

Comparaison : `eq  lt  le  gt  ge  ne`

Mots-clés : equal, less than, less or equal, greater than,...

La comparaison se fait selon l'ordre lexicographique.

Exemples

```
$chant = 'tra' . "la"x3 ; # $chant="tralalala";
if ("la" le $chant) {
  print "OK !\n" ;
}                          # Affiche  : OK !
```

Exercices

1. Proposer au moins 2 méthodes différentes d'afficher la concaténation de 2 chaînes `$a1` et `$a2`.
2. Écrire le programme "Hello world !" en Perl en utilisant une variable pour chaque mot.
3. Que se passe-t-il quand on incrémente une variable inexistante ? Quand on l'affiche ? Expérimenter.
4. Qu'affichent les instructions suivantes ?

```
$a = 1;
$b = "2+$a";
$a = 2;
print $b;
```

5. Les expressions suivantes sont-elles vraies ou fausses ?

```
(1 < "un")
(1 lt "un")
```

6. Que donne `print((2+3)x4);` ?

Les tableaux

Qu'est-ce ?

C'est une **variable contenant une liste de scalaires**.

Un nom de tableau débute toujours par `@`.

Écriture littérale de listes

Entre parenthèses, scalaires séparés par des virgules.

```
@a = (1, 2, 3);
@b = (1, "deux", 3);
my @tab = ($a, $une_autre_variable);
my @tableau_vide = ();
```

Astuces

Opérateur `qw` : *quote word*

`qw(Aleph Uqbar Zahir)` équivaut à `("Aleph","Uqbar","Zahir")`.

Opérateur `..` :

`(1 .. 5)` équivaut à `(1,2,3,4,5)`.

Les tableaux : opérations fondamentales

Concaténation de listes

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

On utilise souvent `push` pour l'empilement en fin de tableau.

Taille d'un tableau

```
@a = ( 0 .. 10 );
$derIndice = $#a ;      # $derIndice=10;
$taille = @a ;          # $taille=11;
```

Accès aux éléments d'un tableau

Chaque élément est un **scalaire**, donc pour un tableau `@tab`, on écrit le premier élément `$tab[0]`.

```
my @b = (1, 2, 3, 4);

# lire la premiere case du tableau
$un = $b[0];            # $un=1;

# lire les 2 premieres cases a la fois
($un,$deux) = @b ;      # $un=1; $deux=2;

# modifier la case de rang 2 (la troisieme)
$b[2] = 8 ;             # @b=(1,2,8,4);

# choisir l'element en comptant depuis la fin
$quatre = $b[-1];       # $quatre=4;
```

Exercices

1. Expérimenter l'interpolation de tableaux, c'est-à-dire `"@tab"`, `"$tab[0]"`.
   Quelle est la différence entre `print @a` et `print "@a"` ?
2. Qu'affiche le programme suivant ?

```
@a = (1,9,3,7,5);
@b = (3,2,1);
$b[1] = @a;
print "@b\n";
```

3. Les arguments d'un programme sont dans un tableau `@ARGV`.
   Écrire un programme qui affiche le dernier et le premier de ses arguments.
4. Échanger deux variables scalaires sans utiliser de variable intermédiaire.

Contexte

Que donne `$scalaire=@tableau;` ?

Perl doit évaluer une liste dans un contexte scalaire. Il effectue alors une conversion implicite.
On peut forcer le contexte avec `scalar(...)`.

Principe

Une "fonction" peut renvoyer des valeurs distinctes suivant le contexte.

```
$a = <$filehandle>; # lire une ligne du fichier
@a = <$filehandle>; # lire toutes les lignes du fichier
```

Et réciproquement ? `@tableau=$scalaire;`
`@tableau=($scalaire);`

Exercices (suite)

5. Écrire un programme qui affiche l'argument dont le rang est donné par le dernier argument. Par exemple, `./test.pl 0 1 2 trois quatre 3` devra afficher `trois`.
6. En utilisant la syntaxe `while ($in=<STDIN>) {…}`, saisir des valeurs ligne par ligne jusqu'à apparition d'une ligne vide. Demander alors le numéro de la ligne à afficher parmi celles saisies.
7. Comment extraire un sous-tableau ? Expérimenter. Comment insérer un élément dans un tableau ?

Les tableaux associatifs

Qu'est-ce qu'un *hash* ?

C'est une collection de **clés** scalaires telle qu'à chaque clé est associée une **valeur** scalaire.

Un nom de *hash* débute toujours par %.

Écriture d'un *hash*

On peut le définir de façon globale.

```
%h = ("un"=>1 , "deux"=>2);
%h2 = ( "Grenoble" => 38000,
        "Lyon" => 69000 );
```

Ou le définir séparément pour chaque paire clé/valeur.

```
my %h ;
$h{"un"}=1 ;
$h{deux}=2 ;
```

Hashes : opérations fondamentales

Accéder à un élément

`$hash{"maclé"}` renvoie la valeur **scalaire** associée à `"maclé"`.

```
%notes = ("Hyppolyte"=>12, "Achille"=>7, "Baudoin"=>9);
$eleve = 'Baudoin';
print "La note de $eleve : $notes{$eleve}.\n";
```

Tester un élément : `exists`

Pour tester l'existence d'une variable scalaire, on utilise `if ($a)` ou mieux `if (defined $a)`.

Pour tester l'existence d'une clé dans un *hash*, il faut utiliser `if (exists $h{"maclé"})`.

Effacer un élément : `delete`

```
%notes = ("Hyppolyte"=>12, "Achille"=>7, "Baudoin"=>9);
delete $notes{Achille};
```

Hashes : opérations fondamentales (2)

keys renvoie la liste des clés d'un *hash*.

```
%h = ( "premier" => 1 , "second" => 2 );
@indices = keys(%h);                    # @indices=qw(premier secon
foreach $index (@indices) {
  print "$index => $h{$index}\n";
}
```

values renvoie la liste des valeurs d'un *hash*.

```
%h = ( "premier" => 1 , "second" => 2 );
@val = values(%h);                      # @val=(1,2);
```

Exercices

1. Écrire un programme qui convertisse un chiffre donné en toutes lettres en argument en valeur numérique. Exemple : `./chiffre.pl trois` affichera "3".
2. En utilisant `while ($in=<STDIN>) {…}`, écrire un programme qui saisisse les noms et les notes d'élèves.
3. Qu'affiche le programme suivant ?

```
#!/usr/bin/perl
@keys = sort keys %ENV;
foreach $key (@keys) {
  print "$key = $ENV{$key}\n";
}
```

4. En utilisant la syntaxe `while ($mot=<STDIN>) {…}`, saisir un mot par ligne, et dire si ce mot a déjà été saisi. Modifier ensuite le programme pour afficher après chaque saisie le nombre d'apparitions de chaque mot.

Plan

Structures conditionnelles : `if/unless`

Booléens

Les valeurs fausses sont `0`, `""`, `()` et `undef`.

Syntaxe courante

`if (condition) { faire; faire; ...}`

```
if ($ARGV[0] eq "--verbose") {
  print "Verbeux.\n";
} elsif (-f $ARGV[-1]) {
  print "Le fichier $ARGV[-1] existe.\n";
}
```

Syntaxe *familière*

`faire if (condition);`

```
print "Verbeux.\n" if ($ARGV[0] eq "--verbose");
print "Stop!" unless $OK;
```

Boucles : `while/until`

Syntaxe courante

while (condition) { faire ; faire ; …}

```
until ($a>2) {
  print ++$a ;
}
while ($a) {
  print $a-- ;
}
```

Affichera ? "123321".

Syntaxe *familière*

faire while (condition) ;

```
print ++$a until $a>2 ;
print $a-- while $a ;
```

Boucles : `foreach`

Syntaxe : foreach $variable (@tableau) { ... }

Parcourt tous les éléments d'un tableau.

```
@tab = (1,3,5,7,9,"...");
foreach $k (@tab) {
  print "$k -> ";
}
```

Affichera `1 -> 3 -> 5 -> 7 -> 9 -> ... ->`

Remarques

- Le scalaire utilisé par `foreach` est local à ce bloc.
- Ce scalaire est un **alias** sur l'élément du tableau (modifiable).

```
@a = (1..5);
foreach $i (@a) { $i *= 3; }
print "@a";
```

Affichera "3 6 9 12 15".

Variables implicites

Perl utilise des variables par défaut, généralement `$_` et `@_`, quand aucune variable n'est précisée.

```
foreach my $i (@tableau) {
  print $i;
}
```

Peut être abrégé en :

```
foreach (@tableau) {
  print;
}
```

ou même :

```
print foreach (@tableau);
```

Quelques fonctions n'utilisent pas `$_` par défaut.
Par exemple, `shift` s'applique toujours aux arguments :

- `@ARGV` dans le corps du programme,
- `@_` dans une une fonction.

Exercices

1. Écrire un programme qui affiche ses arguments, un par ligne.
2. En utilisant `$in=<STDIN>` pour saisir les valeurs, programmer le jeu de devinette d'un entier fixé entre 1 et 99.
3. Afficher la somme des arguments du programme en écrivant l'opération. Par exemple, `./somme.pl 1 2 3 4` affichera 1+2+3+4=10.
4. Écrire un programme qui affiche ses arguments accompagnés de leur nombre de caractères (utiliser la documentation).

Compléments

last

Quitte la boucle en cours.

```
foreach $arg (@ARGV) {
  last if ($arg eq "--quit");
  print "$arg\n";
}
```

next

Passe à l'itération suivante de la boucle en cours.

```
foreach $k (@tab) {
  next unless $hash{$k};
  $k += $hash{$k};
}
```

Compléments (2)

Opérateurs `or` et `and`

Ce sont les mêmes opérateurs que `||` et `&&` mais avec une priorité minimale.

```
$arg = shift
      or die "Le programme a besoin d'un argument";
```

Perl n'évalue que la partie gauche d'une expression si cela suffit à en donner la valeur.

Affectation par défaut

Une syntaxe fréquente : $||=$ et $//=$

```
use strict;
my ($x, $y);
$x = 0;
$x ||= 'vide'; # x faux => $x = 'vide'
$y //= 'vide'; # y undef => $y = 'vide'
```

Debug : la chasse aux erreurs

Les mêmes principes qu'ailleurs s'appliquent :

- indenter et documenter son code,
- le modulariser,
- vérifier les codes de retour des appels de fonctions, etc.
- test unitaires ?

Mais si on a tout de même une erreur :

- Utiliser "perl -w" ou `use warnings;`
- `use diagnostics;`
- `use Data::Dumper; warn Dumper(\@mavariable);`
- `use strict;`
- "perl -d script.pl" : le debugger

Plan

Définition

On utilise le mot-clé `sub`.

```
sub fonction {
   ...
}
```

La fonction est appelée par `fonction()`.

La fonction peut retourner un argument avec `return`.

```
#!/usr/bin/perl -w
sub true {
  return "Vrai";
}
print true();
```

Arguments et variables locales

Une variable locale est crée par l'instruction `my`.

Les arguments sont passés par **références** (et non par copie) dans le tableau spécial `@_`.

Lecture des arguments

Exemple de fonction préservant ses arguments :

```
%h = fonction("texte",3);
sub fonction {
  my ($arg1, $arg2) = @_ ;
  # ...
}
```

Les arguments `$_[0]` et `$_[1]` sont ici copiés dans 2 variables locales.

`shift(@_)` est souvent utilisé pour lire (et dépiler) un argument.

Arguments et variables locales (2)

Toute modification des éléments de `@_` modifie les arguments qui doivent donc être des variables.

```
sub modifie {
  $_[0] .= "\n";
}
modifie("Un texte constant");   # ERREUR
modifie($a);                    # OK
```

Il appartient à la fonction de choisir si elle copie ses arguments, ou si elle les modifie directement comme ci-dessous.

```
sub upcase_array {
  foreach $txt (@_) { $txt = uc($txt); }
}
@a = qw(conte bruit fureur);
upcase_array(@a);          # @a = qw(CONTE BRUIT FUREUR);
```

Exercices

1. Pourquoi ne peut-on pas recevoir comme arguments deux tableaux distincts ?
2. Écrire une fonction qui affiche "Vrai" si son argument est vrai, faux sinon. Proposer une variante syntaxique. Tester avec `(2x3-1<13**2)`.
3. Que fait la fonction suivante ?

```
sub fonction {
  my @array = @_ ;
  foreach (@array) { $_++ ; }
  return @array ;
}
```

4. Que fait la fonction suivante ?

```
sub fonction {
  foreach (@_) { $_*=2 ; }
}
```

Plan

Affichage

print

Affiche ses arguments sur la sortie courante.

```
print 'Je me cache ', $ici, " ou ${la}.";
```

printf

Idem que pour le C.

```
printf "La racine de %d est environ %5f.\n", 3, sqrt(3);
```

warn et die

Affiche un texte sur le canal d'erreur (puis quitte).

Si l'argument ne se termine pas par \n, insère un warning ad hoc.

```
die "Le programme a besoin d'un argument.\n" unless @ARGV;
warn "Is there a bug here?" if $debug;
```

Chaînes de caractères

chop et chomp

`chop` retire le dernier caractère d'une chaîne.

`chomp` retire le retour à la ligne final, s'il y en a un.

Exemple :

```
$saisie = <STDIN>;
chomp($saisie);
```

Attention : écrire **chomp($saisie);**

et non `$saisie=chomp($saisie);`

Sous-chaînes : substr

Syntaxe : substr TEXTE,POSITION,TAILLE

```
$t="Un texte long";
$extrait = substr $t, 3, 5;     # $extrait = "texte";
substr($t,3,5)="commentaire";   # $t="Un commentaire long",
```

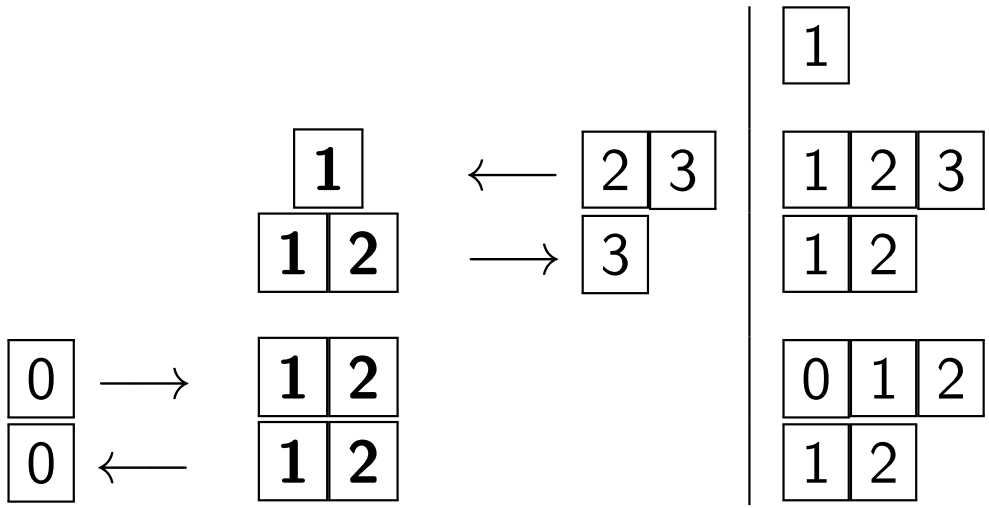
Tableaux (1)

Piles

```
@a=(1);

push @a, (2,3);
$last=pop @a;

unshift @a, (0);
$first=shift @a;
```



Inverser une liste

reverse renvoie la liste dans l'ordre inverse.

```
@a = reverse (1 .. 10);
```

Trier une liste

Par défaut, **sort** trie selon l'ordre lexicographique.

```
@a = sort (glob "*");
```

`@a` contient la liste triée des fichiers du répertoire courant.

Tableaux (2)

Tableaux et chaînes de caractères

join fusionne une liste en une chaîne.

Syntaxe : `join TEXTE,LISTE`

```
@a = qw(un deux trois);
$t = join(" ",@a);          # $t = "un deux trois";  $t="@a"
$t = join(" - ",@a);        # $t = "un - deux - trois"
```

split découpe une chaîne en une liste.

Syntaxe : `split REGEXP,TEXTE`

```
$t = "un - deux - trois";
@a = split / - /,$t;        # @a = ("un","deux","trois")
@b = split //,"lettres";    # @b = qw(l e t t r e s)
```

Exercices

1. Lire des nombres passés en argument au programme et en afficher la liste triée du plus grand au plus petit sous la forme "25 > 17 > 12 > 9 > 5 > 2".
2. Calculer l'intersection de deux tableaux.

```
@a = (1..20);
@b = (-1,2,5,7,22,18);
...
@intersection = (2,5,7,18);
```

3. Calculer le produit scalaire de 2 vecteurs. On lira les vecteurs sous la forme $v_0, v_1, v_2, \ldots$
4. Lister les clés d'un *hash* associées à une valeur donnée en argument.
   Exemple :

```
%h = ( "un"=>1, "deux"=>2, "Un"=>1 , "aussi"=>1);
```

`./clefs.pl 1` devra afficher `'aussi' 'Un' 'un'`.

Plan

Ouvrir un fichier

Syntaxe : `open($FILEHANDLE,"nomdefichier");`

Remarques :

- Par défaut, ouverture en lecture.
- Avant Perl v5.8, on écrivait FILEHANDLE (non scalaire).

Droits d'accès

Syntaxe : `open($FILEHANDLE, "permissions", "fichier");`

Si le nom de fichier est précédé de `<`, `>`, `>>`, `>+`, le fichier est ouvert resp. en lecture, écriture, ajout, lecture/écriture.

```
open $LOG, "<", "/var/log/dmesg"
  or die "Erreur d'ouverture de dmesg : $!";
# ... acces au fichier ...
close $LOG;
```

`STDIN`, `STDOUT` et `STDERR` sont des descripteurs de fichiers ouverts par défaut.

Accès en lecture aux fichiers

opérateur <FILEHANDLE>

Chaque accès scalaire lit une ligne et passe à la suivante.

```
while ($ligne = <$FILE>) {
  print ++$k, ": ", $ligne;
}
```

Le retour à la ligne fait partie de la ligne lue (cf `chomp`).

read

Syntaxe : read FILEHANDLE,SCALAIRE,TAILLE,OFFSET
read FILEHANDLE,SCALAIRE,TAILLE

Renvoie le nombre de caractères lus.

```
open $FILE, "fichier" or die;
read $FILE, $length, 1;
read $FILE, $pstring, $length;
close $FILE;
```

Accès en écriture aux fichiers

print, printf

Par défaut, Perl écrit sur STDOUT.

Pour écrire dans le fichier pointé par `FILEHANDLE`, il faut utiliser `print FILEHANDLE "...";`.

Exemple d'écriture dans un fichier.

```
open $FILE, ">", "fichier" or die $!;
print STDERR "Ecriture en cours...";
print $FILE "Contenu du fichier :\nPas grand-chose.\n";
close $FILE;
```

L'opérateur diamant : <>

Un *filehandle* vide agit comme un filtre.

```
#!/usr/bin/perl -w
while ($txt = <>) {
  print uc($txt);
}
```

L'opérateur <> lit depuis

- les fichiers en arguments, s'ils existent,
- l'entrée courante, sinon.

Alors sont équivalents

- `./diamant.pl fichier.txt`
- `cat fichier.txt |./diamant.pl`

Et `./diamant.pl` sans argument attendra une saisie utilisateur.

Exercices

1. Lire `/etc/passwd` et produire un hash `%root` contenant des clés "home", "shell", "UID", "GID". Interroger ce hash.
2. Écrire un filtre qui compte le nombre de lignes et le nombre de caractères.
3. Écrire un programme qui copie un fichier vers un autre, ligne par ligne.
4. Écrire une procédure de sauvegarde de hash et une procédure de lecture de hash dans un fichier.
5. Écrire un programme qui affiche la fréquence des mots dans un fichier.
6. Construire une fonction `multiprint($texte, $filehandles...)` qui écrive un texte dans plusieurs fichiers.

Exemple complet d'E/S

Exemple complet d'E/S : copie de fichier

```
#!/usr/bin/perl
use strict;
use warnings;
my ($from, $to) = @ARGV;
open $FROM, "<", $from or die $!;
open $TO  , ">", $to   or die $!;
my $line;
while ($line = <$FROM>) {
  print $TO $line;
}
```

Plan

Documentation

Manuel Perl

Quick Start

```
man perlrequick
```

Tutoriel

```
man perlretut
```

Résumé

```
man perlreref
```

Syntaxe des expressions régulières

```
man perlre
```

Opérateurs Perl, section regexp

```
man perlop
```

FAQ spéciale regexp

```
man perlfaq6
```

Exemple

Comment définir un motif qui permette de reconnaître une adresse électronique ?
Plus précisément, comment savoir si un scalaire `$email` est une adresse email.

Critère 1 : `$email` doit contenir un @.
En général, une expression régulière s'écrit entre deux `/`, donc

```
/@/
```

Critère 2 : plus précisément, elle doit contenir @XXX.XX.
Un caractère quelconque s'écrit `.`, donc

```
/@...\.../
```

Critère 3 : le nombre de caractères qui suit @ n'est pas fixé.
Un caractère est supposé répété à loisir s'il est suivi de `+`, donc

```
/.+@.+\..+/
```

Exemple (suite)

Critère 4 : les caractères utilisés sont des lettres minuscules.
Une liste de caractères est donnée par `[abcde]`, abrégée en `[a-e]`, donc

```
/[a-z]+@[a-z]+\.[a-z]+/
```

Critère 5 : il ne doit pas y avoir d'autres caractères avant ou après.
Un `^` initial et un `$` final marquent les extrémités de la chaîne.

```
/^[a-z]+@[a-z]+\.[a-z]+$/
```

Ce qui donne que `$email` doit

- Commencer... (`^`)
- par une ou plus lettres minuscules (`[a-z]+`)
- suivies d'un arobase (`@`)
- suivi de lettres (`[a-z]+`)
- suivies d'un point (`\.`)
- suivi de lettres (`[a-z]+`)
- que rien ne suit (`$`)

Éléments d'une *regexp*

Éléments simples

Beaucoup de caractères se désignent eux-même ("A", "b", "5").
Le "`.`" désigne un caractère quelconque.

Exemples :

```
/abc/    # le texte contient "abc"
/a.c/    # le texte contient "a", un caractere, "c"
```

Classe de caractères

Un caractère parmi une liste s'écrit `[...]`.

Exemples :

```
/[abcd]/     # le texte contient "a" ou "b" ou "c" ou "d"
/[a-d]/      # idem
/[a-zA-Z]/   # le texte contient une lettre ascii
```

Éléments d'une *regexp* (2)

Classe de caractères

Pour une négation de classe, il faut placer un `^` en début de liste.

Exemples :

```
/a[^b]c/     # rejette "abc" et accepte "aac", "a0c", "balcon"
/[a-z][^0-9]/ # rejette "l33t" et accepte "1eet"
```

Classes prédéfinies

`.`		Tout caractère
`\d`	[0-9]	Chiffres
`\w`	[a-zA-Z0-9_]	Alphanumériques (et _)
`\s`	[\r\t\n\f]	Espaces

On peut combiner : `/[\d\s_-]/`

Les négations existent :`\D`, `\W`, et `\S`.

Éléments d'une *regexp* (3)

Multiplicateurs

Combien de fois doit apparaître un caractère ?

`?`	0 ou 1 répétition de la classe de caractère précédant
`*`	0 ou + répétition de la classe de caractère précédant
`+`	1 ou + répétition de la classe de caractère précédant
`{`*i*`,`*j*`}`	*i* à *j* répétition de la classe de caractère précédant
	(par défaut, si *j* est absent, $j = \infty$)

Exemples

	exemples de textes correspondants
`/tions?$/;`	"intuition", "mentions", "ration"
`/^\s*a+/;`	" a", "aaa", " ab"
`/^1.{0,4}2$/;`	"12", "1abcd2", "1aaa2", "1a !2"

Exercices

1. Que donnent les regexps suivantes sur les textes en vis-à-vis ?

`/a.[^c]/`	"pacte", "barque !", "talc", "archaïque"
`/e.+u$/`	"perdure", "feu", "e-sudoku", "repu"
`/\w+\s?\?/`	"Ano- ?", "Oui ?", "No !", "Errr ?"
`/^#!\s*\S+$/`	"# !/bin/sh", "# !/usr/bin/perl -w"

Interpolation dans une *regexp*

Variables

Un `$` qui n'est ni protégé par `\` ni placé en fin d'expression indique un nom de variable.

```
$a='[a-z]\d';
/$a$/;          # idem /[a-z]\d$/
```

Le motif est parcouru à la façon de "...".

```
/\n\t\e/      # newline, puis tabulation, puis escape
/\Q\n\t\E\e/ # \n\t puis escape

$motif="a";
/\U$motif\E/; # /A/
```

Matching : m/.../

Comment appliquer une *regexp* ?

On utilise l'opérateur =~.

```
if ($txt =~ /a/) {
  print "\$txt contient un 'a'.\n";
}
```

Il y a d'autres types de regexp. Pour mieux préciser qu'il s'agit de *matching*, on peut écrire : `$txt =~ m/a/`

Une *regexp* sans opérateur est appliquée à $_.

```
foreach (@tableau) {
  print if /\n$/;
}
```

Exercices

1. Écrire un programme qui reçoit en argument une regexp de matching et teste cette regexp sur chaque ligne lue.
2. Proposer et tester, à l'aide du script 1, des regexp qui reconnaissent
   - une ligne vide.
   - une ligne vide, en dehors d'éventuels espaces.
   - une phrase (majuscule initiale et ponctuation finale).
   - au moins un "a" suivi d'au plus un "s".
3. Écrire une fonction qui teste si une chaîne est un entier naturel, un réel, ou n'est pas un nombre.
4. Tester si une chaîne est valide comme (ancien) numéro d'immatriculation de véhicule.

Substitution : s/*motif*/*remplacement*/

Le second membre remplace la correspondance obtenue par le motif.

Application avec =~ comme pour le *matching*.

Exemples

```
$_ = "Lapin";
s/a/u/;         # "Lupin"
s/^./R/;        # "Rupin"
s/[A-Z]..//;    # "pin"
s/^/Sa/;        # "Sapin"
```

La gourmandise

Quand on applique un multiplicateur, il est gourmand (*greedy*) par défaut : il capte autant de caractères que possible.

```
$_ = "aaabbb";
s/b+/B/;          # $_ = "aaaB";
s/.*/!/;          # $_ = "!";
```

Les multiplicateurs peuvent être non-gourmands (sobres ?) s'ils sont suivis d'un "?" : on obtient *?, +?, et {i,j}?.

```
$_ = "aaabbb";
s/b+?/B/;         # $_ = "aaaBbb";
s/.*?/!/;         # $_ = "!aaaBbb";
```

Exercices

1. Proposer et tester une substitution qui
   - efface le premier caractère
   - transforme "barbare" en "tartare".
2. Écrire une fonction qui supprime les blancs de début et de fin de ligne pour le texte passé en argument.
3. Remplacer "oui" par "non" quelle que soit la casse du premier.
4. Écrire un filtre qui reçoit en argument une regexp de substitution et l'applique à chaque ligne. On utilisera la fonction `eval(". . .")`.
5. Écrire une variante de l'exercice 4 qui reçoive une chaîne (et non un motif) à remplacer et sa substitution. Par exemple, `"a.b"` et `"XXX"` transformera `"arba.b"` en `"arbXXX"`.

Modificateurs

```
$_ = "Karamazov";
s/a/o/;         # "Koramazov" !!!
s/a/o/g;        # "Koromozov"
```

Principaux modificateurs

- **g** global (toutes les occurrences)
- **i** insensible à la casse
- **m** multiligne (^ et $ pour le début et la fin de chaque ligne)
- **s** *single-line* (tout le texte considéré comme une seule ligne)

```
while (m/lapin/g) {
  print "Encore un lapin !\n";
}

s/\n\n+/\n/gs;  # efface les retours chariot multiples
```

Groupes

Alternatives

Comment avoir le choix dans une *regexp* ?

`/chaud|froid/` est identique à `/chaud/ or /froid/`.

`/apostats?|apocalypses?/` correspond à 4 mots.

Groupes

Les parenthèses `()` permettent de regrouper des termes.

```
/apo(stat|calypse)s?/
/(pa)?radis/            # /paradis|radis/
/ar(br|me)(e|ment)/     # arbre, armee, arbrment, armement
```

Mémoire

Chaque correspondance à un groupe est stockée dans une variable `$1`, puis `$2`, etc.

```
s/apo(stat|calypse)s?/$1/    # remplacera apostat par stat
s/(anti|pro)-nucleaire(s?)/$1-centrale$2/
```

Traduction : tr/.../.../

Remplace caractère par caractère.
Renvoie le nombre de remplacements.

```
$_ = "Philistins";

tr/Pi/Co/;
print;          # Cholostons

tr/a-zA-Z/A-Za-z/;
print;          # cHOLOSTONS
```

Découper une chaîne : split

split est bien plus que la fonction réciproque de join.

Syntaxe : split REGEXP,TEXTE

```
$txt = "a=>b => c ===> d";
@termes = split /\s*=+>\s*/,$txt;
print "@termes\n";     # a b c d
```

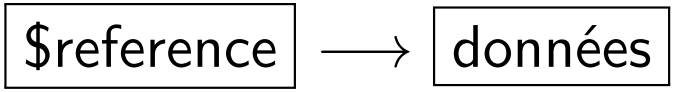
Exercices

1. Proposer et tester une regexp qui
   - échange les deux premiers mots du texte
   - insère une espace (s'il n'y en a pas) avant les signes " ; : ? !"
   - convertit une date ISO au format français (par exemple de 2005-12-30 à 30 décembre 2005)
   - enlève les accents dans un texte
2. Programmer un filtre qui compte le nombre de signes de ponctuation.
3. Écrire un filtre comptabilisant le nombre d'occurrences d'un motif passé en argument.
4. Convertir toutes les balises HTML d'un fichier en majuscules.

Plan

Références

Une référence est un scalaire. C'est en quelque sorte un pointeur *à la Perl*.



Références à des scalaires

Préfixer par un \ référence la variable scalaire.

```
$t="Ahem" ;                    @a=(1,2,0);
$rt=\$t ;                      $ra1=\$a[1];
```

Déréférencement

Préfixer par un $ déréférence le scalaire.

```
print $$rt ;          # print "Ahem"; ($$rt idem $t)
$$ra1 = 1 ;           # $a[1]=1;
```

Pour un scalaire : $reference ⟶ $scalaire=$$reference

Références à des tableaux

De façon identique, `\@a` référence le tableau `@a`.

```
$ra1 = \@array;
foreach $x (@$ra1) { ... }  # idem : foreach $x (@array)
```

ATTENTION : "\" s'applique à une **variable** !

En effet : `\($a,@b)` $\leftrightarrow$ `(\$a,\@b)`.

Tableaux anonymes

Les crochets `[]` construisent directement une telle référence.

```
$ra2 = [1, 2, [9,8,7], $ra1];
```

Déréférencement des éléments avec ->

```
print $$ra2[0];          # 1        (idem ${$ra2}[0])
print $ra2->[1];         # 2        (notation)
print $ra2->[2]->[0];    # 9
print $ra2->[2][1];      # 8        (simplification)
print "@{$ra2->[2]}";    # 9 8 7
```

Références à des *hashes*

À partir d'une variable, avec `\%hash`.
Directement (*hash anonyme*), avec `$rh = { ...}`.

```
$rhash = {
    "Russie" => "Moscou",
    "Japon"  => "Tōkyō  ",
};
foreach $pays (keys %$rhash) {
    print "$pays : Capitale $rhash->{$pays}\n";
}
```

C'est cette syntaxe qui permet d'écrire des enregistrements (les "struct" du C).

Utilisation de références

Passage d'arguments à une fonction

Comment créer une fonction qui renvoie l'intersection de 2 tableaux ?

```
sub intersecte {
   (@a,@b) = @_ ;       # ERREUR !!!
   if ($a[0] == $b[0]) {
```

Il faut passer des références aux tableaux.

```
sub intersecte {
   ($a,$b) = @_ ;
   if ($a->[0] == $b->[0]) {
...
}

@inter = intersecte(\@tab1,\@tab2);
```

Exercices

1. Comparer `\@a` et `[@a]`. Commenter alors le code suivant :

```
@a = (1,2,3);
$r1 = [@a];
push @$r1, 4;
$r2 = \@a;
shift @$r2;
```

2. Écrire une fonction qui lit deux tableaux et renvoie les éléments du premier que ne sont pas dans le second.
3. Écrire un programme qui saisisse ligne par ligne un tableau à deux dimensions, puis demande quelle ligne afficher.
4. Reprendre l'exemple précédent et contruire une variable qui stocke les lignes et les colonnes du tableau.
5. Stocker dans un hash anonyme les informations de /etc/passwd et y accéder à la demande (en choisissant parmi les logins, puis parmi les champs *uid*, *gid*, *home*, *shell*).

Plan

Utilisation d'un module

Exemple

```
#!/usr/bin/perl -w
use LWP::Simple;
$content = get("http://perl.org/");
die "Erreur web?" unless defined $content;
print "$1\n" if ($content =~ m/(Copyright[\s\d-]+)/);
```

Télécharge la page web grâce à la fonction `get()` du module `LWP::Simple`, et affiche son copyright.

Principe

La directive `use Module;` déclare l'utilisation du module.
Un `man Module` est fortement recommandé.

Ce `use` est traité à la compilation, pas à l'exécution.

Le module Getopt

```perl
#! /usr/bin/perl -w
use Getopt::Long qw(:config bundling);

# initialisation des options : valeurs par defaut
%opts = ( verbose => 0,
^^I   debug    => 0   );
# lecture des options
GetOptions(\%opts,
^^I   "help|h",      # "script.pl --help" ou "-h"
^^I   "verbose|v+", # $opts{verbose}
^^I   "configfile|config|c=s");

foreach (@ARGV) {
    if (-d) { push @directories,$_; next; }
    die "Argument non compris : $_\n";
}
```

Modules à syntaxe objet

Certains modules sont seulement objet, d'autres laissent le choix.

```
use XML::Simple;
my $hashref = XMLin("dico.xml");
foreach $k (keys %$hashref) {
  $hashref->{$k} =~ s/\s+$//;
}
XMLout($hashref, OutputFile => "dico_2.xml");
```

Et en version objet :

```
use XML::Simple;
my $obj = new XML::Simple;
my $hashref = $obj->XMLin("dico.xml");
foreach $k (keys %$hashref) {
  $hashref->{$k} =~ s/\s+$//;
}
$obj->XMLout($hashref, OutputFile => "dico_2.xml");
```

Trouver le module souhaité

Distribution spécifique

Debian : 1668 paquets dans la distribution Debian Lenny.

Windows : 7870 modules dans ActivePerl.

CPAN

`cpan.org` centralise des milliers de modules.

Exemples de modules utiles

```
Data ::Dumper # pour savoir ce que contient une variable
Getopt ::Long # pour traiter facilement les arguments et options
Pod ::Usage # pour documenter son programme
XML ::Simple # pour les manips simples en XML
Template     # pour remplir des templates en Perl
File ::Find # pour naviguer dans l'arborescence
LWP ::Simple # pour les acces web simples
```

Exercices

1. Utiliser `XML::Simple` pour sauvegarder et charger un hash dans un fichier.
2. Écrire un programme qui affiche les titres du Monde en ligne (www.lemonde.fr) en utilisant le module `LWP::Simple`.
3. On souhaite que les éléments d'un hash respectent l'ordre dans lequel ils ont été créés. Touver la méthode dans la doc de Perl, puis la mettre en place à l'aide de CPAN.

Inclusion de fichiers de code

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

require "include.pl";
print $truc;
printdate();
```

script.pl

```perl
$truc = "chose\n";
sub printdate {
  # ...
}
1;
```

include.pl

Remarques

- `require "filename"` cherche le fichier dans les chemins du tableau `@INC`. Pour le modifier : `push @INC, "/my/path"`.
- Les répétitions de `require` sont ignorées.
- Tout fichier inclus se termine par `1;`

Inconvénient

Risque de collusions de variables ou de fonctions !

Packages et espaces de nommage

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

require "include.pl";
print $Machin::truc;
Machin::printdate();
```

script.pl

```perl
package Machin;
$truc = "chose\n";
sub printdate {
}
1;
```

include.pl

Remarques

- Par défaut le code est dans `package main;`
- Pour accéder à une variable globale masquée :

```perl
$variable = 5;
sub fonction {
  my $variable;
  print $main::variable;
}
```

Portée des variables

Que se passe-t-il si on passe en `use strict` ?
Il faut déclarer les variables !

Dans un fichier inclus :

- `my` ne peut pas sortir du fichier de déclaration.
- `our` déclare une variable globale :

```perl
#!/usr/bin/perl
use strict;
require "include.pl";
print $Machin::truc;
Machin::printdate();
```

script.pl

```perl
package Machin;
our $truc = "chose\n";
sub printdate {
}
1;
```

include.pl

Modules : importation

Un module peut ajouter des symboles dans `main::`.

```
#!/usr/bin/perl
use English; # charge le module English.pm
use Carp qw(croak); # importe seulement croak()
use Benchmark (); # pas d'import
croak("no"); # importé dans main::
Benchmark::timethese( ... ); # pas dans main::
```

`use` versus `require`

- `use` est lancé lors de la compilation, `require` à l'exécution.
- `use` peut importer des symboles.
- `require Module` charge le module, mais sans import de symboles.

Attention :

`require Term::Readline` $\Longleftrightarrow$ `require "Term/Readline.pm"`

Créer un module

```
package Mon::Module;  # fichier Mon/Module.pm

use strict;
use Exporter;
our @ISA = qw(Exporter);

# variable globale accessible par $Mon::Module::VERSION
our $VERSION = 0.1;

# symboles exportés automatiquement
our @EXPORT    = qw(&fonction1 &fonction2);
# symboles exportables à la demande
our @EXPORT_OK = qw($var1 %hash1 &fonction3);

...

END { ... } # destructeur global (en fin de programme)

1;
```

Plan

Interfaces utilisateurs

- Curses (mode texte)
- Tk (interface historique)
- Gtk (linux) / wxWidgets (wxPerl)

Exemple de Perl/Tk

```
use strict;
use Tk;
my $mainWin = new MainWindow ( -title => 'Hello' );
$mainWin->Label( -text => 'Hello, world!' )->pack();
MainLoop();
```

```
use Tk;
use Tk::LabEntry;
$main = new MainWindow();
my $dial=$main->LabEntry(-width => 50,
        -label=>"RegExp :", -textvariable => \$op);
$dial->pack(-fill => 'x', -expand => 0);
$main->Button(-text => 'Renommer',
```

Perl et le système de fichiers

Lister les fichiers

`glob` renvoie une liste de fichiers correspondant à un motif shell.

```
@header_files = glob "*.h";
```

Tests de fichiers

La syntaxe générale est `-x "nomdefichier"`. Quelques opérateurs :

-e	existence	-x	droits d'exécution	-s	taille en octets
-f	fichier	-r	droits en lecture	-M	âge (en jours)
-d	répertoire	-w	droits en écriture	-f -x	($\geq$ v5.10) ...

```
if ($arg and -d $arg) {
  foreach my $file (glob "$arg/*.pl") {
    print "Perl file: $file\n";
  }
}
```

Exécuter un programme externe

`system` exécute et attend la fin du programme appelé.

```
system("ls -1sh");              # passe par un shell
system("ls", "-1", "-sh");      # sans shell
```

Les *backquotes* renvoient le texte produit par une commande.

```
$fichiers = `ls -1`;
@files = split /\n/,$fichiers;
```

Attention : Ce n'est ni fiable, ni portable.

Rediriger la sortie du programme avec `open`.

```
open LS, "ls -1|" or die "ERREUR $!";
my @files = <LS>;
```

Exercices

1. Afficher tous les fichiers exécutables de `/bin/`, triés selon leur taille. Puis optimiser pour ne demander la taille qu'une seule fois par fichier.
2. Écrire un programme qui change les droits d'accès pour toutes les entrées dans un répertoire donné.
3. Reprendre le programme précédent pour permettre de donner des droits différents aux fichiers et aux répertoires.
4. Ajouter une option récursive au programme précédent grâce à `File::Find`.

Les bases de données

Le module **DBI** (DataBase Independant for Perl) permet l'accès aux bases de données.

La syntaxe est commune à toutes les bases de données acceptées par DBI : MySQL, PostgreSL, Oracle...
Attention : la compatibilité du SQL n'est pas garantie !

Documentation

- `perldoc DBI` pour l'utilisation générale.
- `DBD::mysql` pour le "driver" MySQL.
- `DBD::SQLite` pour le "driver" SQLite, etc.

Exemple MySQL

```
use DBI;
my $DB_NAME     = "produits";
my $DB_USER     = "user";
my $DB_PASSWD   = "password";
my $dbh = DBI->connect(
        "DBI:mysql:$DB_NAME", $DB_USER, $DB_PASSWD)
  or die "Erreur lors de la liaison avec la DB : $!";

$req = $dbh->prepare("SELECT foo, bar ".
                     "FROM table WHERE baz=?");
$req->execute( $baz );
while ( @row = $req->fetchrow_array ) {
   print join(" | ", @row), "\n";
}
$dbh->disconnect;
```

Exercices

1. Utiliser le module `DBI` avec le driver SQLite (`DBD::SQLite`) pour créer une base *base* et une table *passwd*. Remplir cette dernière avec le contenu de /etc/passwd.
   Le SQL nécessaire :

```
CREATE TABLE passwd
    ( login, x, uid, gid, description, home, shell )
INSERT INTO passwd VALUES (...)
SELECT * FROM passwd WHERE ...
```

Plan

Quelques thèmes non abordés

- le débuggeur Perl (`perl -d`)
- la syntaxe étendue des expressions rationnelles
- l'interfaçage avec d'autre langages
- la gestion des processus (Poe)
- la programmation objet (Moose)
- les monolignes (les options CLI avec `man perlrun`)
- le format de documentation POD
- Les nouveautés de Perl 5.10 sq. (`man perl5100delta`)

Et beaucoup de détails : opérateurs, variables prédéfinies, ...

Bibliographie choisie

- **Introduction à Perl**, alias *Le lama*.
  286 pages, 4e édition, 2006. Éditions O'Reilly France.
  Traduction de **Learning Perl** de Schwartz, Phoenix, Foy.
- **Perl moderne**, de Bruhat et al.
  464 pages, 1re édition, 2010. Éditions Pearson.
- **Perl pour l'impatient** de Desreux, Tougard.
  128 pages, 2e édition, 2005. Éditions H&K.

En anglais :

- **Perl Cookbook** de Tom Christansen.
  976 pages, 2nd edition, 2003. O'Reilly.
- **Modern Perl** (en ligne, licence Creative Commons)
  http://www.onyxneon.com/books/modern_perl/index.html
- **Beginning Perl**, publié en 2000.
  http://www.perl.org/books/beginning-perl/

Obfuscative code

Exécutable en Perl et en postscript.

```
/;{}def/#{def}def/$_={/Times-Bold exch selectfont}#/_{rmoveto}#/"{dup}#/*/!/$
;/q{exch}#/x; {/J q #}#/.{/T q #}#{stringwidth}#{}#{}# 14 string dup dup dup
260 40 moveto 90 rotate; %/}};$0='"\e[7m \e[0m"';@ARGV=split//,reverse
q(ThePerl). q(Journal) x 220; q; 0 T putinterval exch 7 J putinterval;
; $_= q /m$ pop T($* !$"=!$ " )pop " * true%? $ " $!" "   !!   !! % !" !"     !
! charpath {!"""}pop $ pop{""!}pop! neg{!#}pop 220! neg _{!!}pop J false %T
charpath  clip " pop 0 " moveto 6{!!}pop $_= 105{!!}pop {$! $ " !  #! ##}
pop{dup dup $ ! " pop pop q{"}pop 22{dup show}repeat {"}pop q 22 mul{$ "} pop
neg{!#! $ "}pop! 8 .65 mul{$ # # $}pop! neg{"}pop  _ pop{"}pop } repeat pop
" {  $ "  !  !  ! $ " ! !" "#"  #"!"""""! #" " # "m/;@ARGV=(@ARGV[-14..-1])x50;q}
 0 "%};s/m[ou]|[-\dA-ln-z.\n_{}]|\$_=//gx;s/(.)(?{$*=''})/('$*.='.(++$#
%2?'':"$0;").'pop;')x(ord($1)-31).'$*'/gee;s/((.(\e\[.m)*|.){77})/$1\n/g;print
; sub showpage {}
```

Japh

Mode lancée par Randal L. Schwartz pour ses signatures dans les newsgroups.

```
$,=" ";print +("hacker,","Just","Perl","another")[1,3,2,0];

print grep(s/\d//,
      sort(split "8hacker, 4Perl 1Just 2another"));

$_="krJhruaesrltre c a cnp,ohet";
      $_.=$1,print$2 while s/(..)(.)//;

$_='x"Not ";"x\"another \";\'x\\"perl \\";x\\"hacker,\\"\'"';
      s/x/print/g;eval eval eval;

$_ = "wftedskaebjgdpjgidbsmnjgc";
tr/a-z/oh, turtleneck Phrase Jar!/; print;
```

Perl golf

Principe

Résoudre un problème avec le programme le plus court possible.

Exemple 1

Convertir le nombre donné en argument de base 36 en base 10.

```
map$.=36*$.-55+/\d/*7+ord,pop=~/./g;print$..$/
```

Exemple 2

Afficher l'ensemble de Cantor au rang donné en argument ;

```
s/./$& $&/g for($\="-")x pop;print
```

Exécution avec les arguments 1, puis 2, puis 3.

```
- -
- -  - -
- -  - -      - -  - -
```

Informations utiles

Pour garder le contact :

francois.gannaz@silecs.info

Les documents utilisés sont disponibles en ligne :

http://silecs.info/formations/Perl/

- Transparents
- Corrections des exercices
- Documents de référence

Licence