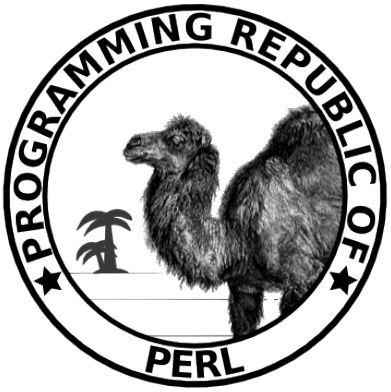
Introduction (rapide) à Perl



Luc DIDRY
Julien VAUBOURG

LP ASRALL
Année universitaire 2009-2010

Université Nancy 2
IUT Nancy-Charlemagne

Table des matières

1 Qu'est-ce que Perl ? 2

2 Document de base 2

3 Déclarer une variable scalaire 2

4 Tableaux 2

5 Tables de hachage (tableaux associatifs) 4

6 Les structures de contrôle 4

7 Les fonctions 5

8 Expressions régulières 5

9 Consulter l'entrée standard 7

10 Manipulation de fichiers 7

11 Fonctions diverses 8

12 La magie de Perl 9

13 Plus de Perl 10

1 Qu'est-ce que Perl ?

Perl a été créé par Larry Wall au milieu des années 80 parce que *awk* montrait ses limites dans le travail qu'il voulait effectuer.

L'interpréteur de Perl compile et exécute ensuite le programme en une seule étape. C'est pourquoi les messages d'erreur comportent souvent « `Execution of programme aborted due to compilation errors.` »

L'orthographe Perl (avec une majuscule) est le plus souvent employée pour parler du langage tandis que perl fera référence à l'interpréteur.

2 Document de base

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

Les lignes autres que le *shebang*, sont appelées des *pragmas*. Ce sont des indications données au compilateur, lui précisant quelque chose à propos du code.

`use warnings` permet d'obtenir des avertissements de la part de perl lorsqu'il rencontre des éléments suspects dans le programme. Ces avertissements ne modifient pas le déroulement du programme mis à part quelques plaintes de temps en temps.

Perl est un langage extrêmement permissif mais l'emploi du *pragma* `use strict` permet de s'imposer une certaine discipline (déclaration préalable des variables entre autres), ce qui permet souvent d'avoir un code plus compréhensible et plus efficace.

Dans les extraits de code de ce polycopié, les deux *pragmas* sont positionnés.

3 Déclarer une variable scalaire[1]

En Perl, les variables ne sont pas typées.

Par défaut, en Perl, toutes les variables sont globales, il est donc possible d'y accéder depuis tout endroit du programme. On peut créer des variables locales en les déclarant avec `my`.

Cependant, l'usage du *pragma* `strict` nous force à déclarer toutes les variables avec `my`. Une variable ne sera donc globale que si on prend le soin de la déclarer en dehors de toute boucle ou sous-programme.

```
my $var = "string";
my $var2 = 42;
my $var3;
```

4 Tableaux

4.1 Déclarer

Comme les variables scalaires, les tableaux se déclarent avec `my` à cause de l'emploi du *pragma* `strict`.

[1] Une variable scalaire est le nom d'un emplacement ne contenant qu'une seule valeur, au contraire, par exemple, des tableaux ou des tables de hachage.

On peut donner des valeurs à un tableau de deux manières différentes : soit en passant les paramètres séparés par des virgules (et en mettant les chaînes de caractères entre guillemets), soit en utilisant `qw` qui permet de s'affranchir des guillemets et des virgules. Par contre si une chaîne contient une espace, il faudra remettre les guillemets.

```
my @tab = ("valeur", 42, "autre");
my @tab2 = qw(valeur 42 autre);
```

4.2 Accéder

```
print $tab[0]; # Attention : c'est bien un $ pour accéder à l'
   élément du tableau
$tab[0] = 42;
print @tab;    # Affichage du contenu : "valeur 42 autre"
print "@tab";  # Affichage de la taille du tableau : 3
```

On voit ici que @tab ne renvoit pas forcément la même chose selon le contexte dans lequel on l'utilise. On parle ici de contexte de liste et contexte de scalaire[2].

Perl renverra automatiquement la valeur nécessaire :

```
$nombre = 42 + @tab; # Contexte de scalaire : 42 + 3 = 45
@copie_tab = @tab;   # Contexte de liste : on recopie "valeur
   42 autre" dans @copie_tab
```

4.3 Les opérateurs `pop`, `push`, `shift` et `unshift`

Les deux premiers permettent de manipuler aisément les tableaux par leur fin :

- `pop` renvoie la valeur du dernier élément du tableau et supprime ce dernier élément

```
$last = pop(@tab); # $last vaut "autre" et @tab vaut "valeur
   42"
pop(@tab);         # On se contente de supprimer le dernier
   élément
```

- `push` ajoute au contraire des éléments au tableau.

```
push(@tab, "nouveau"); # @tab vaut "valeur 42 nouveau"
```

Les deux derniers manipulent les tableaux par leur début :

- `shift` renvoie la valeur du premier élément du tableau et supprime ce premier élément

```
@tab = qw(valeur 42 autre);
$first = shift(@tab); # $first vaut "valeur" et @tab vaut "42
   autre"
shift(@tab);          # On se contente de supprimer le premier
   élément
```

- `unshift` ajoute au contraire des éléments au tableau.

```
unshift(@tab, "nouveau"); # @tab vaut "nouveau 42 autre"
```

[2] On l'a vu précédemment, le scalaire c'est une seule valeur, la liste c'est logiquement plusieurs valeurs.

5 Tables de hachage (tableaux associatifs)

Une table de hachage est une structure de données comme un tableau, en cela qu'elle peut contenir un nombre quelconque de valeurs et les retrouver à la demande. Cependant, au lieu de repérer les valeurs par un indice *numérique*, comme avec les tableaux, elles sont repérées par un *nom*.

5.1 Déclarer

```
my %hash = (
"cle1" => "valeur",
"cle2" => 42,
"cle3" => "autre",
);
```

Il existe d'autres manières d'affecter des valeurs à une table de hachage mais celle-ci a l'avantage d'être la plus lisible.

5.2 Accéder

```
print $hash{"cle1"};          # Affiche "valeur"
$hash{"cle1"} = 42;           # Affectation
@tab_cles = keys %hash;       # @tab_cles vaut "cle1 cle2 cle3"
@tab_valeurs = values %hash; # @tab_valeurs vaut "42 42 autre"
```

6 Les structures de contrôle

Les structures **if**, **if else**, **if elsif**, **while**, **do while** fonctionnent comme dans la plupart des langages de programmation.

Mais comme Perl, c'est trop la classe, on peut faire des raccourcis marrants qui font gagner du temps[3].

6.1 La boucle for

Elle est semblable à tous les autres langages usuels.

```
for(my $i = 0; $i <= 42; $i++) {
    print "$i\n"; # Attention : print n'effectue pas de retour
       a la ligne automatiquement
}
```

6.2 La boucle de parcours foreach

`foreach` permet de parcourir tout un tableau rapidement et sans se préoccuper de sa taille.

```
# Tableau
foreach my $element (@tab) {
```

[3] Et le temps c'est de l'argent. De plus, l'argent c'est le nerf de la guerre dans ce bas monde.

```
    print "$element\n";
}

# Tableau particulier (suite de nombres)
foreach my $i (0..42) { # Pour $i de 0 à 42, inclus
    print "$i\n";   # Affichage identique que celui de la
       boucle for vue ci-avant
}

# Tables de hachage (tableaux associatifs)
foreach my $cle (keys(%hash)) {
    print "$hash{$cle}\n"; # Affiche chaque valeur de la table
       de hachage
}
```

Le second exemple démontre qu'il est aussi possible de remplir un tableau avec les nombres de `n` à `m` de façon automatisée :

```
my @tab = (n..m);
```

7 Les fonctions

7.1 Déclaration

```
sub division {
    # Premier argument divise par le second
    return $_[0] / $_[1];
}
```

Il est aussi possible de définir une ligne permettant de donner aux arguments des noms plus explicites :

```
sub division {
    my ($operande1, $operande2) = @_;
    return $operande1 / $operande2;
}
```

7.2 Appel

On utilise l'esperluette `&` pour indiquer qu'il s'agit d'une fonction (on pourrait s'en passer dans certaines circonstances).

```
print &division(42, 10); # Affiche 4.2
```

8 Expressions régulières

Perl est particulièrement réputé pour sa force sur le traitement des expressions régulières.

Pour la syntaxe des expressions régulières, vous pouvez aller sur http://sylvain.lhullier.org/publications/intro_perl/chapitre10.html (vous y trouverez aussi des exercices).

8.1 Recherche

L'expression régulière recherchée est appelée « motif ». On place le motif entre slashes et on utilise `=~` pour indiquer dans quelle variable on doit rechercher le motif. La recherche de motif renverra `true` ou `false` dans un contexte de scalaire.

```
if($foo =~ /bar/) {
    print "" bar " a ete trouve dans \$foo";
}

if($foo =~ /bar/i) {
    print "" bar ", " bAr " ou " BAR " a ete trouve dans \$foo"
        ;
}
```

Le `i` collé à l'arrière du motif permet de faire une recherche insensible a la casse.

8.2 Substitution

```
my $foobar = "foobarbar";
$foobar =~ s/bar/foo/;
print $foobar; # Affiche: foofoobar

$foobar = "foobarbar";
$foobar =~ s/bar/foo/g; # le " g " permet de ne pas limiter la
   substitution a la premiere occurence
print $foobar; # Affiche: foofoofoo (comme le college)
```

On peut également cumuler les options `g` et `i` en `gi`. Il en existe d'autres (pour plus amples informations, RTFM).

Sachez aussi qu'un certain nombre de caractères doivent être échappés par un anti-slash (ex : / . ? * ...).

8.3 Les classes de caractères

Les raccourcis suivants peuvent être utiles[4] :

- **\d** : Représente tous les chiffres (équivalent à **[:digit:]** ou **[0-9]**)
- **\w** : Représente tous les caractères alphanumériques plus l'underscore, sans les accents (équivalent à [a-zA-Z0-9_])
- **\s** : Représente tous les caractères d'espacement (espace, passage à la ligne, tabulation, saut de page, retour chariot : équivalent à [\n\t\f\r])

8.4 Extraction de sous-chaînes

Allez, parce que Perl est trop puissant, on en remet une couche !

[4] C'est faux : ils ne **peuvent** pas être utiles, ils le **sont** !

```
my $url = "http://www.cpan.org/foobar/";
my ($site, $dossier) = $url =~ /http:\/\/www.(\w+).org/(\w+)\/;
```

9 Consulter l'entrée standard

L'entrée standard, c'est le plus souvent le clavier mais ça peut aussi être un fichier (si vous faites `$ ./mon_prog < mon_fichier` par exemple).

```
while(defined(my $foo = <STDIN>)) { print "$foo\n"; }
```

Cette syntaxe demande à l'utilisateur d'entrer du texte au clavier. Celui-ci peut cesser de fournir les entrées par Ctrl+D. `$foo` contiendra à chaque fois ce qu'a tapé l'utilisateur avant de valider par Entrée. S'il s'agissait d'un fichier, `$foo` contiendra les lignes du fichier, l'une après l'autre.

On utilise `chomp` pour supprimer le retour à la ligne qui finalise chaque ligne de texte (quand l'utilisateur tape sur Entrée pour valider son entrée) :

```
my $foo = "toto\n";
print $foo; # Affiche toto suivi d'un retour a la ligne

chomp($foo);
print $foo; # Affiche toto sans retour a la ligne
```

10 Manipulation de fichiers

10.1 Ouvrir un fichier

Un descripteur de fichier est le nom, dans un programme Perl, d'une connexion d'entrée/sortie entre le processus Perl et le monde extérieur. `STDIN`, vu plus haut, est un descripteur de fichier spécial.

```
my $fichier = "fichier.txt";
open(FICHIER, "<", $fichier) or die("Impossible d'ouvrir le
   fichier $fichier : $!\n"); # Ouverture en lecture seule
open(FICHIER, ">", $fichier) or die( ... ); # Ouverture en
   ecriture (reinitialise le fichier)
open(FICHIER, ">>", $fichier) or die( ... ); # Ouverture en
   ecriture (ajoute a la fin du fichier)

close(FICHIER); # Facultatif (se ferme automatiquement a la fin
    du programme)
```

Il est également possible de d'ouvrir un fichier en lecture/écriture (`+>` : écrasement, `+<` : ajout).

10.2 Consulter un fichier

C'est bien joli d'ouvrir un fichier, encore faut-il s'en servir.

```
while(defined(my $foo = <FICHIER>)) { print "$foo\n"; } #
    Affiche chaque ligne du fichier
```

10.3 Ecrire dans un fichier

```
print FICHIER $foo; # Ecrit $foo dans le fichier ouvert en
    ecriture
```

10.4 Se placer dans un répertoire

Par défaut, Perl se place dans le répertoire d'appel du script. Il cherchera donc, par exemple, les fichiers qu'on lui dit d'ouvrir dans ce répertoire.

```
chdir("/home/asrall");
```

10.5 Explorer un répertoire

L'opérateur **glob** permet l'expansion de nom de fichier exactement comme dans le shell et donc d'explorer le répertoire :

```
my @fichiersPerl = glob("*.pl"); # Remplit un tableau de tous
    les noms de fichiers du repertoire courant correspondant au
    motif
```

11 Fonctions diverses

Rapidement, quelques fonctions que vous aurez sûrement l'occasion d'utiliser :

```
# Si une variable n'a pas ete initialisee, elle vaut undef
if(defined $foo) {
    print "La variable \$foo a ete definie.\n";
} else {
    print "\$foo retourne la valeur undef.\n";
}

if(-d $foo) { print "$foo est un repertoire.\n"; }
if(-e $foo) { print "Le fichier $foo existe.\n"; }
if(-f $foo) { print "$foo est un fichier regulier.\n"; }
# Ces fonctions qui se ressemblent sont les memes que la
    commande test de bash (man test pour toutes les voir)

my @tab = split(/ /, $foo); # Decompose $foo dans le tableau
    @tab d'apres le motif (ici une espace)

my $numerique = int($foo); # Cast $foo en entier

my $aleatoire = rand($max); # Retourne un nombre aleatoire
    entre 0 et $max
```

```
my $enMajuscules = uc($foo); # Retourne $foo en majuscules
my $enMinuscules = lc($foo); # Retourne $foo en minuscules

@foo = sort(@foo); # Trie le tableau @foo selon l'ordre
   asciibetique (1, 10, 2, a, etc.)
@foo = sort { $a <=> $b } @foo; # Trie le tableau selon l'ordre
    alphanumerique (1, 2, 10, a, etc.)
```

12 La magie de Perl

Perl permet d'économiser énormément de caractères, voici quelques raccourcis très utilisés dans la communauté Perl.

12.1 La variable magique

```
for(1..42) {
    print "$_\n";
}

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

En cas d'absence du nom de variable, $_ prend automatiquement le relais.

De plus, en cas d'absence de paramètre, certaines fonctions prennent $_ comme valeur par défaut (c'est le cas ici pour le print du foreach).

12.2 L'opérateur diamant

```
while (<>) {
    print;
}
```

Selon le contexte, le diamant peut représenter deux choses différentes : si un fichier est passé en argument à l'appel du script[5], le diamant le lira ligne à ligne, sinon l'utilisateur sera invité à taper au clavier.

12.3 Les parenthèses

Perl se passe de la plupart des parenthèses des fonctions.
Ces trois lignes sont identiques :

```
chomp($_);
chomp $_;
chomp;
```

[5] Attention, cette fois ça peut être de la forme `$ ./mon_prog < mon_fichier` comme tout à l'heure ou `$ ./mon_prog mon_fichier`.

13 Plus de Perl

Sites :

http://perl.developpez.com/cours/?page=SommaireTutoriels#TutorielsDebuter

http://sylvain.lhullier.org/publications/perl.html

http://articles.mongueurs.net/

Livre :

Introduction à Perl de Randal L. Schwartz, Tom Phoenix et Brian D. Foy
O'Reilly 2006, ISBN : 2-84177-404-X

Les modules additionnels de Perl sont disponibles sur http://search.cpan.org

[6] http ://oreilly.com/pub/a/oreilly/perl/usage/