

Perl
Practical Extraction & Reporting Language

David Odin & Philippe Isorce
CPE Lyon
2015

David Odin & Philippe Isorce (CPE Lyon) Perl 2015 1 / 46

Notes

PERL — PLAN


Présentation

David Odin & Philippe Isorce (CPE Lyon) Perl 2015 2 / 46

Notes

HISTORIQUE

- En 1987, première version par Larry Wall.
- Inspiré des outils **awk**, **sed**, **Shell**, **C**
 - Langage naturel,
 - Riche grâce aux bibliothèques / modules CPAN,
 - Simplifie la réalisation des tâches systèmes.
 - Le couteau suisse des chaînes de caractères.
- Aujourd'hui en version 5.18.2
- Mascotte :



David Odin & Philippe Isorce (CPE Lyon) Perl 2015 4 / 46

Notes

CARACTÉRISTIQUES

- Points positifs :
 - Rapide / efficace pour programmer
 - Grand nombre de possibilités, 21 000 modules
 - Portable d'un système à l'autre
 - Permet l'autodocumentation (perldoc)
- Points négatifs :
 - Difficilement réutilisable, illisible
 - Lent à l'exécution, langage interprété, pré-compilé

Notes

INSTALLATION

- GNU/Linux, par défaut sur
 - Debian depuis la version 2.1
 - Slackware depuis la version 2.5
 - RedHat depuis la version 6.0
 - SuSE depuis la version 6.0
 - Ubuntu, depuis toujours
- Autres Unix
 - Par défaut sur AIX depuis la 4.3.3, sur Solaris 8, sur IRIX
 - Disponible pour HPUX
 - De base sur FreeBSD
 - Mac OSX depuis au moins la version 10.5.
- Windows : existe aussi, mais très déconseillé

Notes

ENRICHIR SON ENVIRONNEMENT PERL

- Beaucoup de modules disponibles par défaut
- Énormément de modules disponibles via le gestionnaire de paquets (`apt-get` pour Debian)
- Installer un module depuis CPAN (en étant root) :

```
perl -MCPAN -e shell
cpan> search Email::Simple
cpan> install Email-Simple
```
- Installer un module source :
télécharger le module source : `Email-Simple-2.003.tar.gz`
extraire l'archive : `tar xvfz module.tar.gz`
`perl Makefile.PL; make; make install`

Notes

DÉBUT D'UN PROGRAMME

- Première ligne de chaque script
Comme pour les scripts bash : `#!/usr/bin/perl -w` (pour warning)
- Utilisation de modules :
 - `use strict`; permet une vérification (recommandé !)
 - `use GD::Graph`;
 - `require Data::Dumper`;
 - beaucoup d'autres sur CPAN

Notes

PERL — PLAN

Spécificités du langage

Notes

BASES SYNTAXIQUES

- Suite de « phrases » terminées par un point-virgule :

```
print "Salut_la_terre";  
# Ceci est un commentaire
```
- Délimitations des chaînes de caractères :

```
$nom = "phil";  
print "Bonjour, _$nom\n";  
Affiche : Bonjour, phil  
print 'Bonjour, _$nom\n';  
Affiche littéralement : Bonjour, $nom
```

Notes

STYLES DE VARIABLES

- Variable Scalaire
 - `$variable` :
 - `$variable` = "Hello_World";
 - `$variable` = 25;
 - `$variable` = 1.234;
- Tableau
 - `@tab`
 - `$tab[0]` première valeur de `@tab`
 - `$tab[$#tab]` dernière valeur de `@tab`
- Table de Hashage
 - `%hash`
 - `%hash{'clef'} = "valeur";`
 - Tableau associatif
- %, @ et \$ sont des Sigils (décorateurs), qui donnent un style à une variable

Notes

EXEMPLES DE VARIABLES SCALAIRES

```
my $animal = "chameau";
print $animal;
print "L'animal_est_un_$animal\n";
my $nval = 42;
print "Le_carre_de_$nval_est_ . $nval*$nval . "\n";
print; # Affiche, par défaut, le contenu de $_
```

Notes

EXEMPLES DE TABLEAUX / LISTES

```
my @animaux = ("loup", "lama", "hibou");
my @nombres = (23, 42, 69);
my @mix = ("chameau", 42, 1.23);

print $animaux[1]; # Affiche "lama"
print $mix[$#mix]; # dernier élément, affiche 1.23
@animaux[0..2]; # donne ("loup", "lama", "hibou");
@animaux[1..$#animaux]; # tous les éléments sauf le premier

my @tri = sort @animaux; # tri
my @maliste = ([ 'toto', 'tutu', 'tata' ],
               [ 'truc', 'much' ],
               [ 'patati', 'patata' ] );
```

Notes

EXEMPLES : TABLES DE HACHAGE

```
my %fruit_couleur = ("pomme", "rouge", "banane", "jaune");
my %fruit_couleur = ("pomme" => "rouge",
                    "banane" => "jaune");
$fruit_couleur->{"pomme"}; # donne "rouge"
my @fruits = keys %fruit_couleur;
my @couleurs = values %fruit_couleur;

# Le Hash contient-il un élément ?
# Test du hash.
if (keys(%nom_du_hash)) {
    # Ici le tableau n'est pas vide
} else {
    # Ici le tableau est vide
}
```

Notes

AUTRE EXEMPLE : TABLE DE HASH

```
# Changer une valeur
my %User;
$User{'tutu'}{'Email'} = 'tutu@parla.fr';
# Ajouter une nouvelle clef
$User{'titi'} = {HomeDir => '/home/titi',
                Passwd => '32fie#$urs3R',
                Email => 'titi@ici.fr' };
# Accéder à la totalité des éléments
foreach my $util (keys %User) {
    foreach (keys %{$User{$util}}) {
        print "$util->$_:$_$User{$util}{$_}\n";
    }
}
```

Notes

RÉFÉRENCES

- Valeur scalaire qui fait référence à un tableau entier ou à une table de hachage.
- Des noms pour les tableaux et les tables de hachage
 - Référence sur un tableau existant :

```
@tableau = (1, "phil", undef, 13);
$href = \@tableau;
```
 - Plus court :

```
$href = [ 1, "phil", undef, 13 ];
```
 - Référence sur un hash :

```
$href = { AVR => 4, AOU => 8 };
```

Notes

EXEMPLE DE RÉFÉRENCES

```
my $variables = {
  scalaire => {
    description => "élément_isolé",
    prefix => '$',
  },
  tableau => {
    description => "liste_ordonnée_d'éléments",
    prefix => '@',
  },
  hash => {
    description => "paire_clef_valeur",
    prefix => '%',
  },
};
print "Les_scalaires_commencent_par_".
      "$variables->{'scalaire'}->{'prefix'}\n";
```

Notes

STRUCTURES DE CONTRÔLE

- Conditions
 - `if (expression) {} else {}`
 - `if (expression) {} elsif (expression) {} else {}`
 - `unless (expression) {}`
 - `$val=(expression ? truevalue : falsevalue);`
- Boucles
 - `do {} while / until (expression);`
 - `for (exp_initiale;test_exp;increm_exp) {}`
 - `while (test_exp) {}`
 - `foreach $i (@liste) {}`

Notes

EXEMPLE : CONDITIONS

```
my $une_condition = 1;
my $a = "foo";
if ($une_condition) {
  my $b = "bar";
  print $a . "\n"; # Affiche "foo"
  print $b . "\n"; # Affiche "bar"
} else {
  print "n'affiche_jamais_rien";
}
print $a; # Affiche "foo"
print $b; # N'affiche rien; $b est hors de portée
print "\n";
```

Notes

EXEMPLE : BOUCLES

```
my $max = 10;
for (my $i=0;$i <= $max; $i++) {
    print $i . ' ' ;
}
print "\n";
my @array = ("bleu", "blanc", "rouge");
foreach (@array) {
    print "L'élément_courant_est_$_\n";
}
my %hash = (1 => "philippe", 2 => "eric", 3 => "david");
# Vous n'êtes pas non plus obligés d'utiliser $_...
foreach my $clef (keys %hash) {
    print "La_valeur_de_$clef_est_$hash{$clef}.\n";
}
```

Notes

CONTRÔLES SPÉCIAUX

- **last** : branchement après la boucle

```
while (exp_1) {
    if (exp_2) { last; }
}
#saut du last ici.
```
- **next** : branchement en fin de boucle

```
while (exp_1) {
    if (exp_2) { next; }
    truc;
    #saut du next ici.
}
```
- **redo** : branchement en début de boucle

```
while (exp_1) {
    #saut du redo ici.
    truc;
    if (exp_2) { redo; }
}
```

Notes

EXEMPLES

```
{
    do {
        last if $x = $y ** 2;
        # faire quelque chose ici
    } while $x++ <= $z;
}
...
for (;;) {
    next if $x == $y;
    last if $x = $y ** 2;
    # faire quelque chose ici
    last unless $x++ <= $z;
}
```

Notes

EXPRESSIONS RATIONNELLES

- Motifs

- `.` : n'importe quel caractère sauf `\n`.
- `\s` : caractère d'espace [`\n\t\f`]. L'inverse est `\S`.
- `\w` : caractères de mots [`a-zA-Z0-9`]. L'inverse est `\W`.
- `\d` : chiffres [`0-9`]. L'inverse est `\D`.
- `/i` : (en fin de motif) permet d'ignorer la casse.
- `[^...]` : tout sauf ce qu'il y a dans les crochets.

- Ancre

- `^` : début de ligne
- `$` : fin de ligne
- `\b` : début ou fin de mot

Notes

EXPRESSIONS RATIONNELLES – SUITE

- Alternatif : `|` correspond à un ou.

- Multiplicatif

- `*` 0 ou n fois, `+` 1 ou n fois, `?` 0 ou 1 fois.

- Utilisation

- `/pattern/option` :
`if ($a =~ /^bon/i)` est vrai si `$a` commence par `bon` ou `Bon`
- `s/pattern/replace/option` :
`$wi =~ s/y/x/g` remplace tous les `y` dans `$wi` par `x`.
- `split` :
`@tab = split(/:/, $v)` coupe la chaîne `$v` suivant le délimiteur `:` et le stocke dans `tab`.

Notes

EXEMPLES D'EXPRESSIONS RATIONNELLES

```
"Hello_World" =~ /world/; # ne correspond pas
"Hello_World" =~ /o W/; # correspond
"Hello_World" =~ /cW/; # ne correspond pas
"Hello_World" =~ /World /; # ne correspond pas
"2+2=4" =~ /2+2/; # pas de correspondance, + est un méta-caractère
"2+2=4" =~ /2\+2/; # correspond, \+ est traité comme un + ordinaire
"The_interval_is_[0,1]." =~ /[0,1\]\./; # correspond
"/usr/bin/perl" =~ /\usr\bin\perl/; # correspond
"housekeeper" =~ /keeper/; # correspond
"housekeeper" =~ /^keeper/; # ne correspond pas
"housekeeper" =~ /keeper$/; # correspond
"housekeeper\n" =~ /keeper$/; # correspond
```

Utilisation de `m`

```
# '/' devient un caractère comme un autre
"Hello_World" =~ m!World!; # correspond, délimiteur '!'
"Hello_World" =~ m{World}; # correspond, notez le couple '{}'
"/usr/bin/perl" =~ m"/perl"; # correspond après '/usr/bin'
```

Notes

GROUPES ET CAPTURES

- Les () permettent de définir des groupes.
- Le cas de correspondance, le contenu des () se retrouve dans les variables \$1, \$2, \$3, \$4...
- Exemple :

```
my $a = "12_avril_1984";
if ($a =~ /(\d+) (.*) 19(\d\d)/)
{
    # $1 contient "12"
    # $2 contient "avril"
    # $3 contient "84"
}
```

- Plus d'info avec : [man perlre](#)

Notes

GESTION FICHIERS

- Descripteurs
 - STDIN entrée standard STDOUT sortie standard
- Fonction open
 - open(FILEH, "monfichier"); ouvre le fichier en lecture
 - open(FILEW, ">monfichier"); ouvre le fichier en écriture
 - open(FILEA, ">>monfichier"); ouvre le fichier en ajout
- Fonction close
 - close(FILEH); ferme le fichier

Notes

GESTION FICHIERS – EXEMPLE

Exemple : copie du fichier exo1 dans exo 2

```
print "Voulez_vous_copier_exo1_dans_exo2_:";
if (<STDIN> =~ /(yes|oui)/i)
{
    my $a;
    open(FILER, "exo1");
    open(FILEW, ">exo2");
    while ($a=<FILER>)
    {
        print FILEW "ligne_de_exo1_:$a";
    }
    close FILER;
    close FILEW;
}
```

Notes

PROGRAMMES EXTERNE

- Exécution de programmes

- `'programme'` sortie sur une chaîne de caractères
- Exemple

```
#!/usr/bin/perl
my @liste='ls';
foreach $_ (@liste) { print "fichier:$_"; }
```

- Redirection de programmes

- `open(FILE, "programme_|");`
- Exemple

```
#!/usr/bin/perl
open(FICHIERGZ, "zcat_fichier.gz|");
while (<FICHIERGZ>)
{
    print $_;
}
close(FICHIERGZ);
```

Notes

FONCTIONS

- Définition de fonctions

```
sub mafonction {
    $_[0]; # premier argument
    @_; # tableau de tous les arguments
    my ($arg1, $arg2) = @_; # notation classique
    ...
    return $var1; # retourne un scalaire
    return $var1, $var2; # retourne plusieurs données
}
```

- Appel de fonction

```
$var = mafonction($first, $second);
@tab = mafonction($first, $second);
&mafonction(@list);
```

- Toutes les variables sont globales

Contournement : `my $variable_locale;`

Notes

FONCTIONS – SUITE

- Exemple

```
#!/usr/bin/perl -w
$a = 1; $b = 2;
$c = sum();
print "Valeur_de_sum:$_$c\nValeur_de_a:$_$a\n";
sub sum {
    my $a = 10;
    return $a+$b;
}
```

- Résultat

```
Valeur de sum : 12
Valeur de a : 1
```

Notes

EXEMPLE DE FONCTION : FACTORIELLE

```
# sous-programme récursif, exemple de factorielle
sub fact
{
  my $n = shift;
  if ($n == 1)
  {
    return $n;
  } else {
    return $n * fact($n - 1);
  }
}
my $nval = 8;
my $result = fact($nval);
print "factorielle_$nval=_,$result\n";
```

Notes

FONCTIONS / OPÉRATEURS CHOP() ET CHOMP()

- Opérateur `chop()` :
 - L'opérateur `chop()` supprime le dernier caractère d'une chaîne, quel qu'il soit. `$b = chop($a)`;
 - Ceci peut présenter un certain risque.
- Opérateur `chomp()` :
 - Ne retire le dernier caractère d'une chaîne que si c'est un `"\n"`
 - Exemple : `$a = <STDIN>`; `chomp($a)`;
 - En plus condensé : `chomp($a = <STDIN>)`;

Notes

ORIENTÉ OBJET

- Classe = Package
 - Objet = Référence
- Exemple :
- ```
package Personne;
my %champs = (nom=>undef, age=>undef, enfants=>undef);
...
use Personne;
$lui = new Personne;
$lui->nom("Dupont");
$lui->age(30);
$lui->enfants(["lucie", "pierre"]);
```
- Méthode = Fonction

Notes

---

---

---

---

---

---

---

---

## EXEMPLE D'UTILISATION D'UN PACKAGE

Personne.pm

```
package Personne;
sub new {#constructeur de la classe
 my $self = {};
 $self->{NOM} = undef;
 $self->{AGE} = undef;
 $self->{ENFANTS} = [];
 bless($self); # voir ci-dessous
 return $self;
}
sub nom { # la méthode de la classe
 my $self = shift;
 if (@_) {
 $self->{NOM} = shift;
 }
 return $self->{NOM};
}
...

use Personne;
my $him = Personne->new();
$him->nom("Philippe");
$him->age(50);
$him->enfants("Matthieu", "Angéline");
push my @All_Recs, $him; # stockage dans
un tableau pour plus tard
printf "l'age de %s est %d ans.\n",
 $him->nom, $him->age;
print "Des enfants sont : ",
 join(", " $him->enfants), "\n";
printf "Dernier nom enregistré %s\n",
 $All_Recs[-1]->nom;
...
```

Notes

---

---

---

---

---

---

---

---

## PERL — PLAN

# Évolution et bibliographie

Notes

---

---

---

---

---

---

---

---

## ÉVOLUTIONS DE PERL

- Perl 5.X évolue toujours
- Perl 6 : réécriture complète, très différents de Perl 4 ou 5.

Notes

---

---

---

---

---

---

---

---

## BIBLIOGRAPHIE

- Introduction à Perl, R. L. Schwartz, T. Phoenix, O'Reilly
- Programmation en Perl, Larry Wall, O'Reilly
- Sites Internet utiles
  - <http://www.perl.org/>
  - <http://www.cpan.org/>
  - <http://perl.enstimac.fr/DocFr>
  - <http://www.mongueurs.net/>
  - <http://perldoc.perl.org/>

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---

Notes

---

---

---

---

---

---

---

---