http://www-adele.imag.fr/users/Didier.Donsez/cours

Le Langage PERL

Practical Extraction and Reporting Language

Didier DONSEZ

Université Joseph Fourier –Grenoble 1

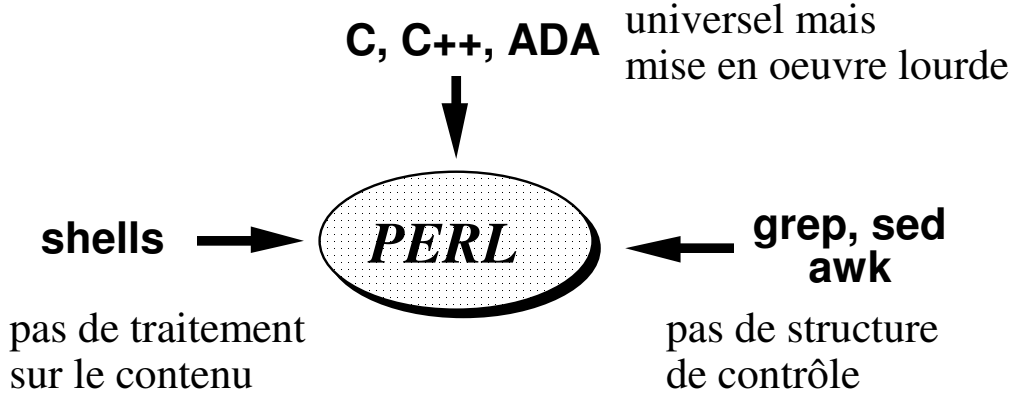
PolyTech'Grenoble LIG/ADELE

Didier.Donsez@imag.fr,

Didier.Donsez@ieee.org

Pourquoi un nouveau langage ?

- Langages de Programmation (C, C++, ADA, Pascal, ...)
  - Manipulation/Traitement de Données Complexes
- Langages de Commande (SH, CSH, ...)
  - Manipulation de Fichiers / Répertoires
  - Lancement de Processus (Commande, Pipeline)
- Utilitaires de Conversion / Formatage (GREP,SED,AWK)
  - Conversion de chaînes (pattern matching)
  - Traitement de Fichiers ligne à ligne (repporting, logging, ...)
  - Fusion des fonctionnalites



- le langage PERL (Larry Wall) Practical Extraction and Reporting Language
  programmation complexe
  manipulation d'expressions régulières
  exécution de processus ...

  Utilisation : Administration système, CGI

Variables Scalaires $var

- Notation
  - `$nom_variable`
  - initialisation par défaut : valeur `undef` (“0” ou chaîne vide)
  - Pas de déclaration (attention au déboggage)
  - Globales par défaut, Locales si spécifiées (local())
  - Variable spéciale `$_` contient le résultat de la dernière expression évaluée. C'est la variable des notations abrégées
- Nombres
  - Littéraux entiers
    - `45  -67`
    - `0377  -0xff`
  - Littéraux flottants (équivalent au double du C)
    - `1.234  +5.6  -78.9`
    - `12.3E4  -0.123e-45`
- Chaînes

```
’coucou’                          # coucou
’c\’est pas gagné!’               # c’est pas gagné!
’A bientôt \n’                    # A bientôt \n
”A bientôt \n”                    # A bientôt suivi d’un
                                  # retour a la ligne
”coucou \177”                     # coucou suivi d’un blanc et d’un delete
`who | grep toto`                 # le résultat de l’execution de la ligne de commande
```

- Interpolation de Variables dans une chaine

```
> afficharg pierre paul
print "@ARGV";                    # pierre paul
print @ARGV;                      # pierrepaul
print '@ARGV\n';                  # @ARGV\n
$var = "appréc"; $variable ="beau !";
print "c'est $variable"           # c'est beau !
print "c'est ${var}iable";        # c'est appreciable
```

Opérateurs

- Nombres
  - opérateurs C : `+ - * / % ++ --`
  - exponentiation `** 2 ** 3 == 8`
  - logique `&& || !`
- Affectations
  - `$a = 3;`
  - `$b = $b * 2;`
  - `$c = 4 + ($a=3);`
  - `$d = $e = 5;`
  - `$b *= 2;`
  - `$b *= $a`
  - `$str .= "world";`
- Fichiers `-r -w -d -f -T ...` `-r $a`

Opérateurs de chaîne

- affectation .
  - $qui = ”world”;
  - $str1 = ”Hello $qui”; # ”hello world”
  - $str2 = ’Hello $qui’; # ”hello $qui”
  - $str = <<IDENTIFICATEUR;
  - plusieurs lignes
  - IDENTIFICATEUR
  - $dochtml = <<_Fin_Doc;
  - <HTML>
  - <HEAD><TITLE>Reponse</TITLE></HEAD>
  - <BODY>
    <A HREF=”http://perl.com”>Des info sur Perl</A>
  - </BODY>
  - </HTML>
  - _Fin_Doc
- concaténation .
  - ”hello” . ”world” # “helloworld”
- répétition x
  - “cou” x 2 # “coucou”
  - “hip ” x (2+1) . “hourra” # “hip hip hip hourra”
- extraction substr
  - substr("hourra", 1, 2) # ==“ou”
- position index
  - index("hourra", "ou") # ==1
- pattern matching
  - $a =˜ /pat/ # VRAI si "pat" est dans $a
  - $a =˜ s/pat/pot/ # substitution des "pat" en "pot" dans $a
  - $a =˜ tr/a-z/A-Z/ # conversion dans $a
- conversions binaires
  - $hex32 = pack("CCCC",140,186,65,25);
  - @valeurs = unpack("C*","bonjour");

Structures de Contrôle

- Bloc d'instruction
  - {
  - inst1;
  - inst2;
  - ... ;
  - derniere_inst;
  - }
- expression de condition
  - chaîne vide ou chaîne "0" — FAUSSE
  - sinon — VRAI
- if / unless
  - if (expr) {
  - inst_si_vrai;
  - ... ;
  - } else {
  - inst_si_fausse;
  - ... ;
  - }

  - if (exp1) {
  - inst_si_exp1_vrai;
  - ... ;
  - } elsif(exp2) {
  - inst_si_exp1_fausse_et_exp2_vrai;
  - ... ;
  - } else {
  - inst_si_exp1_fausse_et_exp2_fausse;
  - ... ;
  - }

  - unless (expr) {
  - inst_si_fausse;
  - ... ;
  - } else {
  - inst_si_vrai;
  - ... ;
  - }

Structures de Contrôle

- while / until
  - while (expr) {
  - inst_tantque_expr_vrai;
  - ... ;
  - }
  - until (expr) {
  - inst_tantque_expr_fausse;
  - ... ;
  - }
- for (comme en C)
  - for (init_expr; test_expr; incr_expr) {
  - inst1;
  - ... ;
  - }
  - init_expr;
  - while (test_expr) {
  - inst1;
  - ... ;
  - incr_expr;
  - }
- foreach
  - foreach $i (@liste) { # $i local a la boucle
  - inst1;
  - ... ;
  - }
  - @val = (2, 5, 9);
  - foreach $n (@val) { # $n référence $val[]
  - $n *= 3;
  - }
  - # on a maintenant @val = (6, 15, 27)

Structures de Contrôle

- last

  sortie de boucle (for, foreach, while) et de blocs indépendants {} (c'est à dire autre que if(){})

```
while( exp1 ) {
  inst1;
  if( exp2 ) {
    inst2;
    last;   # sortie du while
  }
  inst3;
}
# reprise du last
```

- next

```
while( exp1 ) {
  inst1;
  if( exp2 ) {
    inst2;
    next;  # saut de la suite corps du while
  }
  inst3;
# reprise du next
}
```

- redo

```
while( exp1 ) {
# reprise du redo
  inst1;
  if( exp2 ) {
    inst2;
    redo;  # saut au début du corps du while
           # sans réévaluation de exp1
  }
  inst3;
}
```

Structures de Contrôle

- blocs étiquetés
  - last, next et redo sur boucles ou blocs imbriqués

```
EXTERIEUR:
for($i=1; $i<=10; $i++ ) {
  INTERIEUR:
  for($j=1; $j<=10; $j++ ) {
    if($i*$j==63) {
      print “$i fois $j égal 63 ! /n”;
      last EXTERIEUR;
    }
    if($i<=$j) {
      next EXTERIEUR;
    }
  }
}
```

- modificateur d’expression

```
expression if exprtest; # if(exprtest){expression;}
expr while  exprtest; # while (exprtest){expr;}
expr unless exprtest; # unless(exprtest){expr;}
expr until  exprtest; # until (exprtest){expr;}
```

- && , || , ?:

```
ceci && cela;
ceci || cela;
exptest ? expsi : expsinon
```

Listes et Tableaux @

- Représentation Littérale
  (1, 2, 3) ("coucou", 4.5) ()
  (1..3) # (1, 2, 3)
  (3.3 .. 6.1) # (3.3, 4.3, 5.3)
- Variables Tableaux @ et Affectation
  @tab = (1,2,3);
  @TAB = (4,5,6);
  @chiffres = (0, @tab, @TAB, 7, 8, 9);
  ($a,$b,$c) = @tab; # $a=1; $b=2; $c=3;
  ($a,$b) = @TAB; # $a=4; $b=5;
  ($a,@TAB) = @tab; # $a=1; @TAB=(2,3);
  @tab = @TAB = @chiffres;
- Accès aux éléments
  @tab = (1,2,3);
  $a = $tab[0];
  $b = $tab[$a];
  $d = $tab[10]; # $d contient undef
  @tab2 = @tab[0,2]; # @tab2 = (1,3);
  $tab2[3]=7; # @tab2=(1,3,undef,7);
- Bornes
  $#tab représente l'indice supérieur de @tab
  $[tab représente l'indice inférieur de @tab
  $nbElem_tab = $#tab - $[tab + 1;

Listes et Tableaux @

- Opérateurs push() et pop()
  - @tab = (1,2); $a = 3;
  - push(@tab, $a); # @tab=(@tab,$a);
  - push(@tab, 4,5,6); # @tab=(@tab,4,5,6);
  - $b = pop(@tab); # @tab=(1,2,3,4,5); $b=6;
- Opérateurs shift() et unshift()
  - @tab = (2,3); $a = 1;
  - unshift(@tab, $a); # @tab=($a,@tab);
  - unshift(@tab, -1,0); # @tab=(-1,0,@tab);
  - $b = shift(@tab); # @tab=(0,1,2,3); $b=-1;
- Opérateurs reverse() et sort()
  - @tab = (1,3,5,0,2,4);
  - @tab = reverse(@tab); # @tab=(4,2,0,5,3,1);
  - @tab = sort(@tab); # @tab=(0,1,2,3,4,5);
  - @tab = sort {$a > $b} @tab;
  - @tabstr = sort("petit","moyen","grand");
  - # @tabstr = ("grand","moyen","petit");
- Opérateur chop()
  - chop(@tabstr); # @tabstr=("gran","moye","peti");
- Paramètres du programme
  - @ARGV

Tableaux Associatifs %

- Tableaux indexés par des chaînes de caractères
- Affectation

```
%tab = ( "coucou",   " ca va",
         123.4,      567
);
$tab{"salut"}   = " tout baigne ?";
$tab{"coucou"} .= " ?";
$tab{123.4}    += 433;
print $tab{"hello"};     # undef
%copie_de_tab   = %tab;
@liste          = %tab;
```

- Tableaux Associatifs Prédéfinis

```
$homedir   = $ENV{'HOME'};
$SIG{'UP'} = 'IGNORE';
```

- Opérateur keys()

```
foreach $k (keys %tab) { print $k $tab{$k} ; }
```

- Opérateur values()

```
@listevaleur = values(%tab);
```

  - # @listevaleur=(" ca va ?",567," tout baigne ?") ou
  - # @listevaleur=(567," ca va ?"," tout baigne ?") ou

- Opérateur each()

```
while( ($cle, $valeur) = each(%tab)) {
      print $cle." ".$valeur;
}
```

- Opérateur delete()

```
%tab = ("coucou","ca va ?",123.4,567);
@element = delete $tab{"coucou"}; # @element=("coucou","ca va") %tab=(123.4,567)
```

Sous Routines &

- Définition

```
sub nom_sous_routine {
  inst1;
  inst2; ... ;
}
```

- Appel &
- sans paramètre
  - while ($i = &fnc_incr)
  - { ... ; }
- avec paramètre
  - &imprimer("hello","world");
- Résultat
- le résultat d'une sous routine est le résultat de la dernière expression évaluée.

```
sub somme_a_b {
  $a + $b;
}
$a = 2; $b = 3;
$c = &somme_a_b;  # $c == 5
```

- Tableau de Paramètres @_

```
sub somme2 {
  $_[0] + $_[1];
}
$c = &somme2 ( $a, $b );
sub somme
{ $s = 0;
  foreach ( @_ ) { $s += $_ ; }
  $s;  # car retour dernière expression évaluée
}
$c = &somme($a,$b,7,11);
```

- Variables Locales local()

```
sub somme {
  local( $s ) = 0;
  local( $p0, $p1, $p2, $p3 ) = @_ ;
 ... ; }
$s = 1;
$res = &somme(1,2,3,4); # $s == 1
```

Expressions Régulières (i)

- > grep "ab*c" nomfic
  - **while($_ = <>) {**
  - **if($_ =~ /ab*c/){**
  - **print $_; }**
  - }
- *En notation abrégée*

```
while(<>) {
   if( /ab*c/ ){ print; }
}
```

- Motif (Pattern)
  - Simple
    - . tout caractère sauf un newline
    - [abc] un des caractères
    - [a-z] une des minuscules
    - [^0-9] tout caractère sauf un chiffre
    - \d [0-9] et \D [^0-9]
    - \w [a-zA-Z0-9] et \W [^a-zA-Z0-9]
    - \s [\r\n\t\f] et \S [^\r\n\t\f]
- Constructeur
  - séquence abc [a][b][c]
  - répétition a* 0 ou plusieurs a
  - a+ 1 ou plusieurs a aa*
  - a? 0 ou 1 a
  - a{5} exactement 5 a
  - a{5,} 5 ou plus a
  - a{0,5} au maximum 5 a
  - complétion a.*b a[^b]*b
  - alternative coucou|salut
  - a|b|c [abc]
  - mémorisation A(.)B(.)C\2D\1E # AwBxCxDwE VRAI # AwBxCyDzE FAUX
- Ancrage
  - \b extrémité de mot
  - \B non extrémité de mot
  - /fred\b/ # alfred, fred mais pas frederic
  - /\Bfred/ # alfred mais pas fred, frederic
  - /^a/ chaîne commençant par a
  - /a$/ chaîne se terminant par a
- Priorité
  - Parenthèses > Multiplicateur > Séquences et Ancrages > Alternatives

Expressions Régulières (ii)

- Opérateurs =~ et !~
- recherche
  - /abc/ # $_ =~ /abc/
  - $str =~ /abc/
  - $str !~ /abc/
- substitution
  - s/abc/def/ # $_ =~ s/abc/def/
  - VRAI si $str contient abc
  - VRAI si $str NE contient PAS abc
- Interpolation de Variables
  - $strings2search = join("|",@ARGV);
  - while(<>) {
  - if(/$strings2search/) { print; }
  - }
- Variables Spéciales
  - $_ = "il fait beau et chaud !";
  - /il fait (\w+)\W+(\w+)\W+(\w+)/;
  - print 'il fait '. $3 .' '. $2 .' '. $1;
  - ($shiny,$and,$hot) = /il fait (\w+)\W+(\w+)\W+(\w+)/;
- Substitution
  - $_ = "beau et chaud";
  - s/(\w+)\W+(\w+)\W+(\w+)/$3 $2 $1/; # $_="chaud et beau"
  - Option
  - /abc/i ignore majuscules-minuscules
  - /$strings2search/o construction de l'automate 1 fois pour toute
  - while(<>) {
  - if(/$strings2search/o) {
  - $strings2search .= '|' . $tab[i++];
  - # n'est plus pris en compte dans l'automate
  - }}
  - s/abc/def/g remplacement de toutes les occurrences de abc
- Opérateurs split() et join()
  - @liste = split(':',$str);
  - $str = join('###',@liste);
  - $regex = '/#{3}/';
  - ($uname,$epasswd) = split($regex,$str,2);

Entrées / Sorties

- Descripteurs standards
  - STDIN, STDOUT, STDERR
- Entrée STDIN

```
while($_ = <STDIN>) {
 if($_=~/abc/){ print $_; }
}
```

- Sortie STDOUT

```
print;
print $_;
print 3 , "petits cochons";
print 1+2 , "petits cochons";
print (1+2 , "petits cochons");
print (1+2) , "petits cochons"; # erreur syntaxe
printf "le $gml et les %2d %10s", 3, "petits cochons";
```

- Sortie STDERR

```
print STDERR;
print STDERR $_;
print STDERR 3 , "petits cochons";
```

- Opérateur Diamant <>

```
while(<>) {
    if(/abc/){ print; }
}
```

  - pgrepabc fic1 fic2 fic3

```
while(<STDIN>) {
 if(/abc/){ print; }
}
```

Fichiers

- Descripteur
  - DESCRIPTEUR en majuscule
- Ouverture/Fermeture
  - open(DESCR ,"nomfic");
  - open(MONFIC ,"fic"); #ouverture en lecture/écriture
  - open(MONFIC_RD ,"< fic"); #ouverture en écriture
  - open(MONFIC_WR ,"> fic"); #ouverture en écriture
  - open(MONFIC_AP ,">>fic"); #ouverture en écriture à la suite
  - open(MONFIC,"fic") || die ("Peut pas ouvrir !");
  - close(MONFIC); #fermeture
- Descripteurs standards
  - STDIN, STDOUT, STDERR
- Lecture
  - while(<MONFIC>) { # $_ contient une ligne du fichier
  - while($ligne = <MONFIC>) { # $ligne contient une ligne du fichier
- Ecriture
  - print MONFIC "salut\n";
  - print MONFIC "hello\n";
- Ecriture Formatée
  - printf DESCR "format", varval1, ...
  - printf MONFIC "%16s = %f ", $strexpr, $valexpr ;
- Opérateurs de Test
  - -r -w -x -o -e -z -s -f -d -l -M -A -C ...

Fichiers

- Formatage des Sorties

```
format ETIQUETTE =
==============================
| @<<<<<<<<<<<<< @<<<<<<<<<< |
    $nom,         $prenom
| @<<<<<<<<<<<<< Bureau @<<< |
  $service,     $bureau
==============================
.
open(ETIQUETTE,">etiq"); open(EMPLOYE,"empl");
while(<EMPLOYE>) { chop;
  ($nom,$prenom,$service,$bureau) = split(/:/);
  write ETIQUETTE;
}
```

Répertoires et Fichiers

- Changement de Répertoire de Travail
  - chdir("/etc") || die("cd /etc impossible");
- Globbing
  - @listefic = </etc/host*>;
  - while($nomfic = </etc/host*>) {
  - if(-r $nomfic) { print $nomfic; }
  - }
- Manipulation
  - opendir(ETC,"/etc");
  - while($nom = readdir(ETC)) {
  - print $nom; }
  - }
  - closedir(ETC);
- Destruction de Fichier
  - unlink($nomfic);
  - unlink(<*.o *%>);
  - foreach(<*.o *%>) { unlink; }
- Renommage de Fichier
  - rename($oldname,$newname);
  - rename("fic","rep/fic");
- Liens
  - link($nomfic,$lien_physique_fichier);
  - symlink($nomfic,$lien_symbolique_fichier);
- Répertoire
  - mkdir("monrep", 0755); rmdir("monrep");
- Droits
  - chmod(0644,"fic1", $nomfic2);
- Propriété
  - chown($uid,$gid,"fic1", $nomfic2);
- Information
  - stat($nomfic);
  - print "Lisible" if -r _;
  - print "Modifié dans l\'heure" if -M < 1/24.0;

Gestion de Processus

- system()
  - system("date");
  - system("date >ficdate");
  - system("(date;who) >ficwho &");
  - system "grep", "toto", "/etc/passwd";
- backquote ``
  - $date = `date`;
  - foreach(`who`) {
  - ($qui,$ou)=/(\S+)\s+(\S+)/;
  - print "$qui est sur $ou";
  - }
- Pipeline
  - open(WHO, "who |"); # lecture du pipe
  - @who =<WHO>;
  - open(LPR, "| lpr"); # écriture du pipe
  - print LPR $rapport;
  - close(LPR);
- Fork
  - if(fork) { # je suis le fils
  - ...
  - exec("date");
  - }else{ # je suis le père
  - wait;
  - }
- Signaux
  - $SIG{'QUIT'} = 'IGNORE';
  - $SIG{'INT'} = 'trait_sigint';
  - sub trait_sigint {
  - $SIG{'INT'} = 'DEFAULT';
  - $SIG{'QUIT'} = 'DEFAULT';
  - }
  - kill(2, $pidfils1, $pidfils2); # SIGINT == 2
- Variables d'environnement %ENV
  - while(($key,$value) = each %ENV) {
  - print $key . '='. $value .'\n'; }

Les Extensions Perl V5

- Enregistrements

```
$rec = {
        nom  => "Paul",
        age   => 23
        amis => [ "Bill", "Alice", "Bob" ],
    };
print 'les amis de '. $rec->{nom}
       ' sont ' @{$rec->{amis}}
```

- Objets
- Threads

Objets (i)

- Interface
  - use Person;
  - $him = Person->new();
  - $him->name("Jason");
  - $him->age(23);
  - $him->peers("Norbert", "Rhys", "Phineas");
  - push @All_Recs, $him; # save object in array for later
  - printf "%s is %d years old.\n", $him->name, $him->age;
  - print "His peers are: ", join(", ", $him->peers), "\n";
  - printf "Last rec's name is %s\n", $All_Recs[-1]->name;
- Constructeur et Méthodes
  - package Person;
  - use strict;
  - sub new {
  - my $self = {};
  - $self->{NAME}   = undef;
  - $self->{AGE}   = undef;
  - $self->{PEERS} = [];
  - bless($self);   # but see below
  - return $self;
  - }
  - sub name {
  - my $self = shift;
  - if (@_) { $self->{NAME} = shift }
  - return $self->{NAME};
  - }
  - sub age {
  - my $self = shift;
  - if (@_) { $self->{AGE} = shift }
  - return $self->{AGE};
  - }
  - sub peers {
  - my $self = shift;
  - if (@_) { @{ $self->{PEERS} } = @_ }
  - return @{ $self->{PEERS} };
  - }
  - 1; # so the require or use succeeds

Objets (ii)

- Constructeurs

```
$me  = Person->new();
  $him = $me->new();
  sub new {
     my $proto = shift;
     my $class = ref($proto) || $proto;
     my $self  = {};
     $self->{NAME}   = undef;
     $self->{AGE}    = undef;
     $self->{PEERS}  = [];
     bless ($self, $class);
     return $self;
  }
```

- Méthodes

```
sub exclaim {
  my $self = shift;
  return sprintf "Hi, I'm %s, age %d, working with %s",
  $self->{NAME}, $self->{AGE}, join(", ", $self->{PEERS});
}
sub exclaim2 {
  my $self = shift;
  return sprintf "Hi, I'm %s, age %d, working with %s",
  $self->name, $self->age, join(", ", $self->peers);
}
sub happy_birthday {
  my $self = shift;
  return ++$self->{AGE};
}
```

Objets (iii)

- Variables de Classe

```
my $Census = 0;
   sub new {
      my $proto = shift;
      my $class = ref($proto) || $proto;
      my $self  = {};
      $Census++;
      $self->{NAME}   = undef;
      $self->{AGE}    = undef;
      $self->{PEERS}  = [];
      bless ($self, $class);
      return $self;
   }
   sub population {
      return $Census;
   }
sub DESTROY { --$Census }
package Employee;
   use Person;
   @ISA = ("Person");
sub salary {
      my $self = shift;
      if (@_) { $self->{SALARY} = shift }
      return $self->{SALARY};
   }
1;
use Employee
   my $empl = Employee->new();
   $empl->name("Jason");
   $empl->age(23);
   $empl->salary(12000);
   printf "%s is age %d and is paid %d.\n",
         $empl->name, $empl->age, $empl->salary;
```

Objets (iv)

- Surcharge
  - sub peers {
  - my $self = shift;
  - if (@_) { @{ $self->{PEERS} } = @_ }
  - return map { "PEON=\U$_" } @{ $self->{PEERS} };
  - }
  - ...
  - $empl->peers("Peter", "Paul", "Mary");
  - printf "%s\n", join(", ", $empl->peers);
    - His peers are: PEON=PETER, PEON=PAUL, PEON=MARY
  - ...
  - printf "%s\n", join(", ", $empl->SUPER::peers);
    - Peter Paul Mary
  - ...
  - printf "%s\n", join(", ", $empl->Person::peers);
    - Peter Paul Mary

Bibliographie

- Des sites
  - http://perl.com
  - http://www.ActiveState.com
  - http://search.cpan.org/ pour la recherche d'extensions
- Des bouquins
  - Randal L. Schwartz , « Learning Perl », ISBN 1-56592-042-2, O'Reilly
  - Randal L. Schwartz , « Introduction à Perl », ISBN 2-84177-005-2, O'Reilly
    - une introduction a Perl v4 qui existe aussi en Français
  - Tom Christiansen & Nat Torkington, « Perl en action », 1re édition, septembre 1999, ISBN : 2-84177-077-X, O'Reilly
  - Larry Wall, Tom Christiansen & Randal L. Schwartz, « Programming Perl », 1-56592-149-6, Ed O'Reilly.
    - Toutes les fonctionnalites de Perl v5
  - Sriram Srinivasan; Advanced Perl Programming, O'Reilly, First Edition, August 1997, ISBN 1-56592-220-4, 434 pages.
  - David N. Blank-Edelman, Perl for System Administration, O'Reilly, First edition, July 2000, ISBN 1-56592-609-9
  - Perl in a Nutshell, by Ellen Siever, Stephen Spainhour & Nathan Patwardhan
  - Programming Perl, 3rd Edition, by Larry Wall, Tom Christiansen & Jon Orwant
  - Advanced Perl Programming, by Sriram Srinivasan
  - Perl Cookbook, by Tom Christiansen and Nathan Torkington
  - Perl for System Administration, by David N. Blank-Edelman