Une brève introduction à Perl

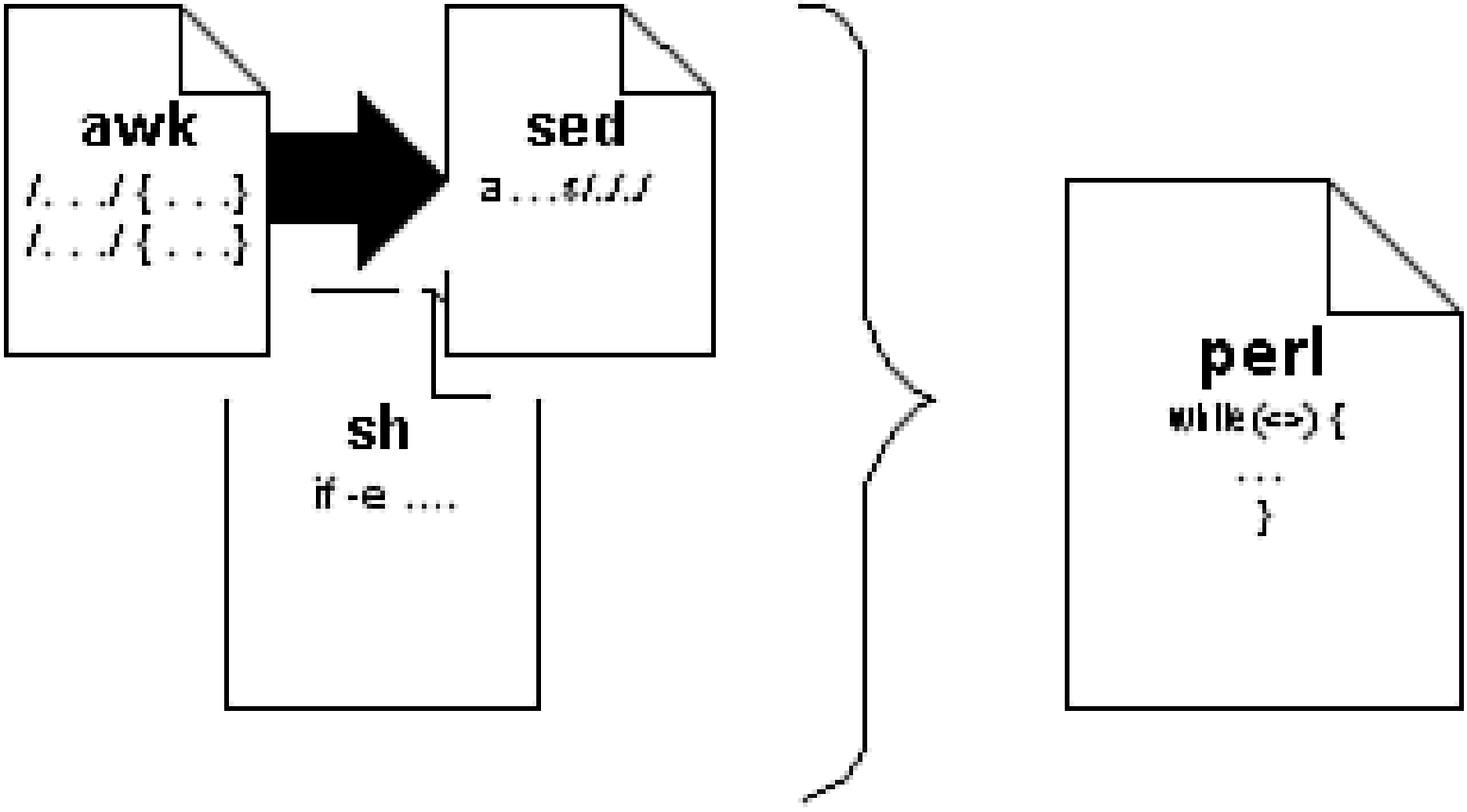
Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay_gu

INF600A
27 novembre 2018

Contenu

1. Introduction : Qu’est-ce que Perl ?


awk
/. . ./ { . . .}
/. . ./ { . . .}
sed
a . . .s/./../
sh
if -e ....
perl
while (<>) {
. . .
}

D'où vient Perl ?

Perl est un langage de programmation créé par Larry Wall en 1987 pour traiter facilement de l'information de type textuel.

Ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C, mais aussi de langages de scripts `sed`, `awk` *et* shell *(*`sh`*).*

Il prend en charge les expressions régulières dans sa syntaxe même, permettant ainsi directement des actions sur l'aspect général de séquences de texte.

Source: *https://fr.wikipedia.org/wiki/Perl_%28langage%29*

D'où vient Perl ?

Perl est un langage de programmation créé par Larry Wall en 1987 pour traiter facilement de l'information de type textuel.

Ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C, mais aussi de langages de scripts `sed`, `awk` *et* shell *(*`sh`*).*

Il prend en charge les expressions régulières dans sa syntaxe même, permettant ainsi directement des actions sur l'aspect général de séquences de texte.

Source: *https://fr.wikipedia.org/wiki/Perl_%28langage%29*

D'où vient Perl ?

Perl est un langage de programmation créé par Larry Wall en 1987 pour traiter facilement de l'information de type textuel.

Ce langage, interprété, s'inspire des structures de contrôle et d'impression du langage C, mais aussi de langages de scripts `sed`, `awk` *et* shell *(*`sh`*).*

Il prend en charge les expressions régulières dans sa syntaxe même, permettant ainsi directement des actions sur l'aspect général de séquences de texte.

Source: *https://fr.wikipedia.org/wiki/Perl_%28langage%29*

Quel était l'objectif visé par Perl ?

Perl is designed to assist the UNIX user with common tasks that are probably too heavy or too portability-sensitive for the shell, and yet too **weird** *or short-lived or complicated to code in C or some other Unix tool language.*

«*Learning Perl*», Schwartz & Wall, 1993

Quel était l'objectif visé par Perl ?

Perl is designed to assist the UNIX user with common tasks that are probably too heavy or too portability-sensitive for the shell, and yet too **weird** *or short-lived or complicated to code in C or some other Unix tool language.*

«*Learning Perl*», Schwartz & Wall, 1993

Que signifie l'acronyme «perl» ?

Perl officially stands for Practical Extraction and Report Language, except when it doesn't.

Perl was originally a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It quickly became a good language for many system management tasks. Over the years, Perl has grown into a general-purpose programming language. It's widely used for everything from quick "one-liners" to full-scale application development.

Que signifie l'acronyme «`perl`» ?

Perl officially stands for Practical Extraction and Report Language, except when it doesn't.

Perl was originally a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It quickly became a good language for many system management tasks. Over the years, Perl has grown into a general-purpose programming language. It's widely used for everything from quick "one-liners" to full-scale application development.

Perl actually stands for Pathologically Eclectic Rubbish Lister, but don't tell anyone I said that.

Source*: Extrait de* `man perl`

Est-ce difficile d'apprendre Perl ?

Perl combines (in the author's opinion, anyway) some of the best features of C, sed, awk, and sh, so **people familiar with those languages should have little difficulty with it.**

Source*:* `man perl`

Quelle est la principale devise de Perl ?

The Perl motto is "There's more than one way to do it." Divining how many more is left as an exercise to the reader.

Quelle est la version la plus récente de Perl ?

Pendant plusieurs années une remarque humoristique perdurait concernant la sortie de **Perl 6** *au sein de la communauté Perl. À la question "Quand le langage Perl 6 sera-t-il disponible ?", la réponse habituelle était "à Noël", mais sans préciser l'année.*

En 2015, c'est-à-dire après quinze ans d'attente, la version dite "de Noël" est finalement annoncée.

Source: `https://fr.wikipedia.org/wiki/Perl_6`

Quelle est la version la plus récente de Perl ?

Pendant plusieurs années une remarque humoristique perdurait concernant la sortie de **Perl 6** *au sein de la communauté Perl. À la question "Quand le langage Perl 6 sera-t-il disponible ?", la réponse habituelle était "à Noël", mais sans préciser l'année.*

En 2015, c'est-à-dire après quinze ans d'attente, la version dite "de Noël" est finalement annoncée.

Source: `https://fr.wikipedia.org/wiki/Perl_6`

Quelle est la version la plus récente de Perl ?

Pendant plusieurs années une remarque humoristique perdurait concernant la sortie de **Perl 6** *au sein de la communauté Perl. À la question "Quand le langage Perl 6 sera-t-il disponible ?", la réponse habituelle était "à Noël", mais sans préciser l'année.*

En 2015, c'est-à-dire après quinze ans d'attente, la version dite "de Noël" est finalement annoncée.

Source: `https://fr.wikipedia.org/wiki/Perl_6`

Note : Depuis Perl 5 (1994), le langage est «orienté-objet», donc avec des objets, références, modules. . . mais çe n'est pas particulièrement «élégant» — on sent le «raboutage».

2. Quelques éléments de base de Perl

Les *sigils*

Les *sigils*

sigil (Ésotérisme) Symbole graphique ou sceau représentant une intention ou un être magique. Source : `https://fr.wiktionary.org/wiki/sigil`

Ruby ⇒ Portée des variables

`foo`	variable locale
`@foo`	variable d'instance
`@@foo`	variable de classe
`$foo`	variable globale

Perl ⇒ Type des variables

`$foo`	scalaire
`@foo`	tableau
`%foo`	hash
`&foo`	sous-routine (pointeur vers, généralement optionnel)
`*foo`	*typeglob* (permet l'aliasing de symboles)

Le préfixe «$» dénote un scalaire


```
$ perl -de1
DB $a = 1;

DB print $a;
1
```

Note : «perl -de1» = Interactions en ligne de commande via le *debugger*

Le préfixe «@» dénote un tableau


```
DB @a = (10, 20, 30);

DB print @a;
102030

DB printf "%d %d %d", @a;
10 20 30


DB print $a[0];
10
```

Remarquez le dernier cas : On utilise «$» puisque la valeur résultante est un scalaire.

Le préfixe «%» dénote un *hash*


```
DB %a = {"abc" => 100, "def" => 200, "xyz" => 990};

DB print %a;
HASH(0x1df6748)

DB print $a["abc"];
100

DB print $a[2];
30

DB print $a;
1
```

Remarquez... la surcharge de `a` !

Définitions et appels de sous-routines

Les parenthèses à l'appel sont optionnelles de même que le `return`


```
DB sub foo { 2 };

DB sub bar { return "bar" }


DB print foo;
2

DB printf foo();
2

DB print bar;
bar
```

Quelques variables spéciales

La variable spéciale `$_`

`$_` variable par défaut — ligne lue dans un fichier, opérande pour *pattern-matching*, etc.

`@_` tableau contenant les arguments dans une sous-routine

```
DB print "abcdef" =~ /a(.*)f/
bcde

DB $_ = "abcdef"

DB print /a(.*)f/
bcde
```

La variable spéciale `@_`

`$_` variable par défaut — ligne lue dans un fichier, opérande pour *pattern-matching*, etc.

`@_` tableau contenant les arguments dans une sous-routine

```
DB sub f { my($x, $y) = @_; print "x: $x; y: $y"; }

DB f "abc", 12;
x: abc; y: 12
```

Donc, comme en Ruby, les parenthèses lors d'un appel à une sous-routine sont optionnelles et comme en Bash il n'y a pas de signature (pas liste des paramètres).

Lecture/traitement de fichiers

Lecture et traitement d'un fichier


```
$ cat foo.txt
abc
def
11
222
```

```
$ perl -de0
DB open( FOO, "< foo.txt" );

DB $x1 = <FOO>; print $x1;
abc

DB $x2 = <FOO>; print $x2;
def

DB while( <FOO> ) { print; }
11
222
```

- `FOO` = Descripteur de fichier ouvert (en lecture) par `open` pour `foo.txt`
- `<FOO>` = Prochaine ligne du fichier associé à `FOO`
- Par défaut : `<FOO>` affecte la ligne lue à `$_` et `print` utilise aussi `$_`

Lecture et traitement d'un fichier… et le bon choix de *sigil*


```
$ cat foo.txt
abc
def
11
222
```

```
$ perl -de0
DB open( FOO, "< foo.txt" );

DB $x = <FOO>; print $x;
abc

DB @x = <FOO>; print @x;
def
11
222
```

- `$x = <FOO>` : Contexte scalaire ⇒ la prochaine ligne est lue et affectée à la variable `$x`, laquelle est ensuite imprimée
- `@x = <FOO>` : Contexte tableau ⇒ toutes les lignes restantes sont lues et affectées au tableau `@x`, lequel est ensuite imprimé

3. Un exemple de script : Calcul de la moyenne des notes et production d'un histogramme

Objectif du programme = Produire l'histogramme des notes pour un travail

Produire l'histogramme des notes pour un travail à partir d'un fichier texte de notes ayant la forme suivante :

```
$  cat INF9876-99.txt
CODE PERMANE : NOM              : DEV1 : INTRA : DEV2 : FINAL :
------------ : -------------    : 20   :  25   : 20   :  35   :
MAXJ05065801 : MAXIMUM * J      : 10.0 :  50.0 : 10.0 :  50.0 :
FPET20118603 : FPERGLIXXI * T   :  9.0 :  46.2 :  9.0 :  46.7 :
FSMQ20029409 : FSMWZIVX * Q     :  9.9 :  49.6 : 10.5 :  43.8 :
GLIV31607906 : GLIR * V         :  9.0 :  47.9 : 11.0 :  34.0 :
...
PIHN25018700 : PIHYG * N        :  9.5 :  44.8 : 11.0 :  43.2 :
QEMH10077306 : QEMKRER * H      :  8.3 :  40.4 :  9.0 :  28.7 :
VELM01017606 : VELQESYM * M     :  9.6 :  35.5 : 10.0 :  25.4 :
VSFX24057409 : VSFIVX  * X      : 10.0 :  44.6 : 10.5 :  32.7 :
WEYQ13119008 : WEYZI * Q        :  9.6 :  42.4 : 10.0 :  43.8 :
WXPN24109108 : WX-PSYMW * N     : 10.0 :  43.6 : 10.0 :  43.3 :
```

Objectif du programme = Produire l'histogramme des notes pour un travail


```
$  ./moyenne.pl INF9876 99 FINAL 5
*** Calcul de la moyenne "FINAL" pour notes-INF5171-99.txt ***
La note maximale possible est   50.0
Nombre d'etudiants = 17
Moyenne = 34.99 (70.0 %)
Note minimale =   17.1
Note maximale =   46.7

La distribution des notes est la suivante:
[ 0-  5]:
( 5- 10]:
(10- 15]:
(15- 20]: *
(20- 25]:
(25- 30]: ****
(30- 35]: *****
(35- 40]: *
(40- 45]: *****
(45- 50]: *
```

```
sub numeroDeNote {
    my ($ligne, $nomNote) = @_;

    my $numNote = 0;
    my @items = split(/$SEPARATEUR/, $ligne);
    my $trouve = 0;
    while ( !$trouve && $numNote <= $#items ) {
        if ( $items[$numNote] =~ /$nomNote/i ) {
            $trouve = 1;
        } else {
            $numNote += 1;
        }
    }
    return $trouve ? $numNote : -1;
}
```

- `my $x` : Définition locale (à la sous-routine) de `x`

  Si «`my`» est omis, alors c'est la variable globale `$x` qui est modifiée — donc comme dans un script *bash*
- `$#x` : Nombre d'éléments du tableau `x`

```
sub noteMax {
    my ($ligneAveMaxs, $numNote) = @_;

    $ligneAvecMaxs =~ /MAXIMUM/ ||
         die( "Il ne semble pas...\n" );

    my @maxs = split(/$SEPARATEUR/, $ligneAvecMaxs);

    return $maxs[$numNote];
}
```

Une autre façon d'écrire le test, dans le style Ruby plutôt que *bash* :

```
die( "...\n" ) unless $ligneAvecMaxs =~ /MAXIMUM/;
```

```perl
sub creerHistogramme {
    my ($numNote, $max) = @_;
    my $total = 0;
    my $nbNotes = 0;
    my $noteMin = $max;
    my $noteMax = 0;
    my @histogramme = ();

    while( <NOTES> ) {  # $_ = prochaine ligne fichier NOTES
        $_ !~ /,/ || die( "Des ',' semblent presentes: A verif
        my @items = split(/$SEPARATEUR/, $_);
        my $note = $items[$numNote];
        $nbNotes += 1;
        $total += $note;
        $note > $max && print "La note pour $items[1] ($note) 
        $noteMin = $note < $noteMin ? $note : $noteMin;
        $noteMax = $note > $noteMax ? $note : $noteMax;
        $numBucket = $note == 0 ? 0 : int(($note-0.01)/$taille
        $histogramme[$numBucket] += 1
    }

    return($total, $nbNotes,  # \@: Reference vers le tableau.
           $noteMin, $noteMax, \@histogramme);
}
```

```
$sigle = $ARGV[0]; # Comme en Ruby: ARGV[0] = 1er argument!
$numGroupe = $ARGV[1];
$nomNote = $ARGV[2];
$tailleBucket = $ARGV[3] ? $ARGV[3] : $TAILLE_BUCKET_DEFAUT;

$nomFichier = "notes-$sigle-$numGroupe.txt";
open(NOTES, "< $nomFichier") || die("Fichier inexistant!\n");
print "*** Calcul de la moyenne \"$nomNote\" pour $nomFichier

$ligneAvecNomsDesNotes = <NOTES>;
$numNote = numeroDeNote($ligneAvecNomsDesNotes, $nomNote);
$numNote >= 0 || die("Ce travail/examen ne semble pas exister!

$ligneAvecMaxs = <NOTES>;
$max = noteMax($ligneAvecMaxs, $numNote);
print "La note maximale possible est $max\n";

my($total, $nbNotes, $noteMin, $noteMax, $histogramme_ref)
  = creerHistogramme($numNote, $max);

$nbNotes > 0 || die("Aucune note n'a ete trouvee!\n");

afficherStats($total, $nbNotes, $max, $noteMin, $noteMax);
afficherHistogramme($histogramme_ref, $max, $tailleBucket);
```

Donc, avec votre connaissance de `bash` et Ruby...

vous devriez être capable de lire des programmes Perl

Donc, avec votre connaissance de `bash` et Ruby. . .

vous devriez être capable de lire des programmes Perl

et si jamais l'envie vous en vient (!?). . .

Donc, avec votre connaissance de `bash` et Ruby. . .

vous devriez être capable de lire des programmes Perl

et si jamais l'envie vous en vient (!?). . .
peut-être même d'en écrire
et de découvrir les joies, et les peines, de Perl ☹

Donc, avec votre connaissance de `bash` et Ruby. . .

vous devriez être capable de lire des programmes Perl

et si jamais l'envie vous en vient (!?). . .
peut-être même d'en écrire
et de découvrir les joies, et les peines, de Perl ☹

- Java
  «*Write once, run everywhere*» (dixit Sun)

Donc, avec votre connaissance de `bash` et Ruby. . .

vous devriez être capable de lire des programmes Perl

et si jamais l'envie vous en vient (!?). . .
peut-être même d'en écrire
et de découvrir les joies, et les peines, de Perl ☹

- Java
  «*Write once, run everywhere*» (dixit Sun)
- Perl
  «*Write once, read never*»

Donc, avec votre connaissance de `bash` et Ruby. . .

vous devriez être capable de lire des programmes Perl

et si jamais l'envie vous en vient (!?). . .
peut-être même d'en écrire
et de découvrir les joies, et les peines, de Perl ☹

- Java
  «*Write once, run everywhere*» (dixit Sun)
- Perl
  «*Write once, read never*»
  «*Write-only* programming»