Langage Perl

Introduction
&
Retour d'expérience

Perl

- *Practical Extraction and Report Langage*
- Langage de script (non compilé)
- Conçu par Larry Wall
- Enrichi par de nombreux développeurs
  - => 18700 packages en 2010
  - DB, web, graphique, analyse de trace apache, fichier excel
  - Domaines spécifiques : ex BioPerl pour la biologie
- Perl5 (1994), Perl5.10 (2009)
  - => compatibilité des programmes depuis 1994

Objectifs du langage

- Combiner les avantages de 3 langages
  - Shell => simplicité, langage interprété
  - C => structures de contrôles et structures de données
  - Awk => **expressions régulières** (manipulation de texte)
- Fonctionnalités supplémentaires
  - + **tableaux associatifs** : tableaux indicé par une chaîne de caractères
  - + **gestion de la mémoire** (automatisée)

Tâche simple : ex1

```
#!/usr/bin/sh
cp f1.txt /local/data
chmod a+r /local/data/f1.txt
```

```
#!/usr/bin/perl
system("cp f1.txt /local/data");
chmod a+r "/local/data/f1.txt";
```

- Pas de compilation
- Pas de déclaration de fonction ni de déclaration de variable
- Très semblable au shell

Tâche simple : ex2

```
#!/usr/bin/sh
for file in *.txt
do
   if [ ! -f test1/$file ]
   do
     cp $file test1
     chmod a+r test1/$file
   done
done
```

```
#!/usr/bin/perl
for $file (<*.txt>)
{
   if (! -f test1/$file)
   {
     system("cp $file test1");
     chmod a+r "test1/$file";
   }
}
```

- Utilisation de structure de contrôle f**or, if , !**
- Et de test sur les fichiers **-f** (vrai si le fichier existe)
- Programmeur shell => perl

Utilisation de packages ou modules

- Tous les modules sont regroupés sur le CPAN
  - *Comprehensive Perl Archive Network*
  - http://www.cpan.org

- Exemple 1: LWP::Simple
  - *LWP = Library World-Wide Web pour Perl*
  - Fonction get retourne le contenu d'un URL

```
#!/usr/bin/perl

use LWP::Simple;
print(get("ftp://ftp.cpan.org/CPAN/README"));
```

Utilisation de package objet

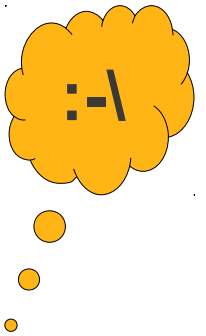
- Exemple 2 : package Net::FTP
- Utilisation avec une syntaxe objet
  - objet = class->new
  - objet->methode

```
#!/usr/bin/perl

use Net::FTP;
$ftp = Net::FTP->new("ftp.cpan.org") or die("$!");
$ftp->login("anonymous",'-anonymous@');
$ftp->cwd("/pub/CPAN/");
$ftp->get("ls-lR.gz");
$ftp->quit();
```

Programmes plus complexes

- ~ programmes C, java etc ...
  - Structures de contrôle
    - if, for, while, next etc ...
  - Fonctions
    - variables locales, globales
  - Structures de données
    - Tableaux, tableaux associatifs
    - Références (pointeurs), objets
  - Expressions régulières
    - Description de chaines de caractères

Expressions régulières



- Permet de décrire (*match*) une chaîne de caractères
  - $variable =~ /expression régulière/

Est-ce que ma variable
- contient une tablutation ?
- contient un ou plusiurs chiffres ?
- commence par le caractère #

- On peut décrire :
  - 1 caractère spécial : **\n \t**
  - Classe de caractères :
    - **\d** (digit) : un chiffre [0-9]
    - **\s** (*space*) = espacement (blanc ou tabulation)
  - Position: **^** = début de chaîne, **$** = fin de chaîne
  - Répétition : ***** = le caractère précédent répété 0 à N fois
    - **A*** = 0 ou plusieurs fois A : A, AA, AAA
    - **\d*** = 0 ou plusieurs chiffres (un nombre) : 1234, 56
  - Alternative : **|** exemple FR | US | UK

```
$var1 =~ /ABC\d*/
```

ABC suivi de 0 ou N chiffres

```
$var2 =~ /FR|UK/
```

Alternative

```
$var3 =~ /^\s*$/
```

Début de chaîne

1 espace

0 ou N fois

Fin de chaîne

Utilité des expressions régulières

- Filtrer certaines lignes d'un fichier
- Control de qualité sur des données
- Vérifier les valeurs saisies par un utilisateur
- *Découper* et vérifier une chaîne en plusieurs parties
  - Adresse mail
    - Vérification : uniquement des lettres, chiffres et quelques symboles autorisés, 1 seule fois @
    - Découpage : user + domaine

Structures de données

- Types de variables
- Nom des variables
- Gestion de la mémoire
- Tableau
- Tableau associatif

3 types de variables

Scalaire
Numérique, chaîne

42	Fichier1.txt	0.374e+08

Tableau
(liste, vecteur)
Suite ordonnée

Indice	Valeur
0	37
1	58
2	3
3	72

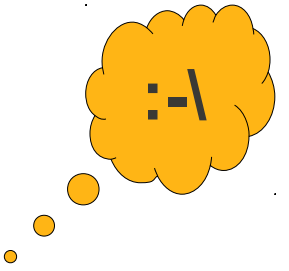
Indice	Valeur
0	503
1	fichier1.txt
2	fichier2.txt
3	fichier3.txt

Tableau associatif
Ensemble de couples (clé, valeur)

Clé	Valeur
rouge	0x00F
vert	0x0F0
bleu	0x00F

Clé	Valeur
position	chr3:1000-1400
score	0.374e+08
annnotation	Gene A

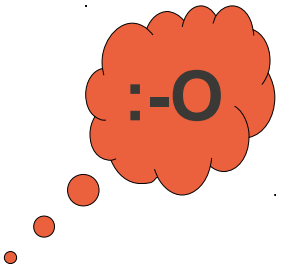
Noms de variables



- Toujours préfixé par un symbole indiquant le type
  - $a : scalaire (chaîne ou numérique)
  - @tab : tableau (indicé par des entiers)
  - %hash : tableau associatif (indicé par des chaînes)

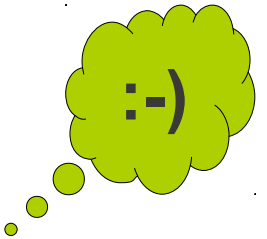
- Utilisation globale
  - $a = "/usr/local/bin";
  - @tab = ("fichier1", "fichier2", "fichier3");
  - %hash = ("rouge" => 0x00F, "vert" => 0x0F0, "bleu" => 0x00F);

- 1 élément d'un tableau est 1 scalaire
  - $i = $tab[2];
  - $j = $hash{"rouge"};



•Attention :
$a, @a, %a = 3 variables différentes

Gestion de la mémoire

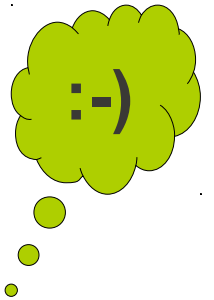


- Allocation et libération de mémoire automatiques

- $tab[3] = "abcd"
  - Allocation d'un tableau de taille 4
  - Tous les éléments contiennent une valeur nulle sauf le 3

- $tab[1000000] = "efgh"
  - Agrandissement automatique du tableau
  - Seuls 2 éléments contiennent une valeur
  - En interne, éventuellement certaines parties du tableau ne sont alloués que lorsqu'on y place une valeur

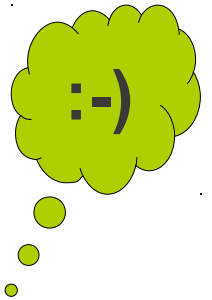
Tableau <=> Liste



- Utilisation avec des indices : classique
  - $tab[1] = 25;
- Utilisation sans indice numérique : liste
  - **push @tab, $element** : *rajouter un élément à la fin*
  - **$element = shift @tab** : *extraire le 1er élément et décaler le tableau*
  - **for $element (@tab) {** : *parcourir les éléments*

```
    $sum += $element;
}
```

Tableau Associatif

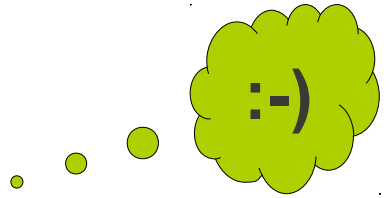


- Ensemble de couples (clé, valeur)
- Permet d'avoir des chaînes comme indice
  - + simples
- Exemple

```
%user_group = ("cecile" => "stat",
               "loic" => "stat");
               "stefan" => "devel",
               "david" => "admin");

$user_group{"sophie"} = "guest";

@user = keys(%user_group);
```

Liste de toutes les clés

Tableau Associatif – exemple 2
Dans un dossier, compter le nombre de fichiers pour chaque type d'extension

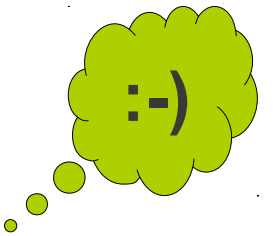


- Utilise directement le nom de l'extension
- Pas besoin de connaître la liste des extensions à l'avance
- Allocation dynamique des éléments du tableau
- Initialisation à 0 => OK pour l'incrémentation

```
#!/usr/bin/perl

for $fichier (<*>) {
    ($nom, $ext) = split(/\./, $fichier);
    $compte{$ext}++;
}
for $ext (sort keys(%compte)) {
    print ".$ext : $compte{$ext}\n";
}
```

```
.csv : 1
.pl : 10
.png : 3
.xls : 2
```

Tableaux associatifs à *N dimensions*




```
for $user ("cecile", "david", "stefan") {
    for $month ("janvier", "fevrier", "mars") {
        $planning{$user, $month} = "busy!";
    }
}
```

```
for $i (1 .. 10) {
    for $j (1 .. 100) {
        for $k (10 .. 20) {
            $result{$i, $j, $k} = mon_calcul($i, $j, $k);
        }
    }
}
```

Package + structure de données

- DBI => Base de données
- Graph => Génération de graphiques
- Apache::ParseLog => Analyse de traces

- Apache::ParseLog + Graph

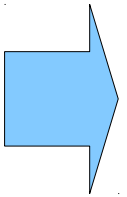
DBI => accès aux bases de données

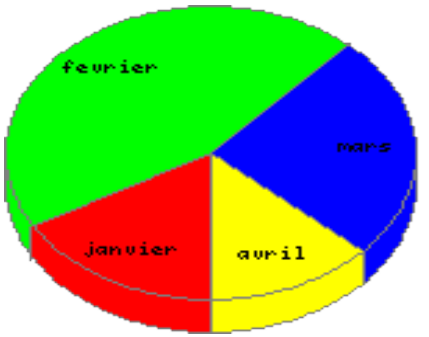
- @row = tableau contenant les champs sélectionnés

```
use DBI;

$db = DBI->connect("dbi:mysql:DB1",'u','p') or die("$DBI::errstr");
$req = $db->prepare("SELECT * FROM table1;");
$req->execute();
while(@row = $req->fetchrow_array()) {
   print join(" ", @row) . "\n";
}
$db->disconnect();
```

Graph => Génération de graphique

@labels	janvier	février	mars	avril
@values	10	27	15	8






```
use GD::Graph;
use GD::Graph::pie ;

@labels = ('janvier', 'février', 'mars', 'avril');
@values = (10, 27, 15, 8);
@data = (\@labels, \@values);

$graph = GD::Graph::pie->new(600, 400) or die($graph->error);
$image = $graph->plot(\@data) or die($graph->error);

open(F1, '>pie.png') or die("Cannot write to pie.png: $!");
print F1 $image->png; # image->jpg image->postcript
close F1;
```

Apache::ParseLog
Analyse de fichier de traces

- %bydate : tableau associatif
- Clé = la date, valeur = nombre d'erreurs
- keys(%bydate) => liste de toutes les clés
- $bydate{$date} : la valeur associée à la clé

%bydate

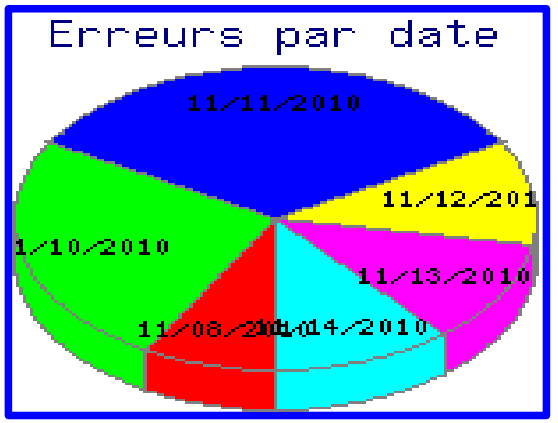
Clé	Valeur
11/14/2010	4
11/15/2010	3
11/16/2010	5
11/17/2010	11
11/19/2010	5
11/20/2010	3

```
use Apache::ParseLog;

$base = new Apache::ParseLog("/etc/apache2/apache2.conf");
$errorlog = $base->getErrorLog();

%bydate = $errorlog->allbydate();
for $date (sort(keys(%bydate))) {
    print "$date:\t$bydate{$date}\n";
}
```

Parser les traces d'erreurs

Comptabilise les erreurs par date

Apache::ParseLog + Graph::Pie



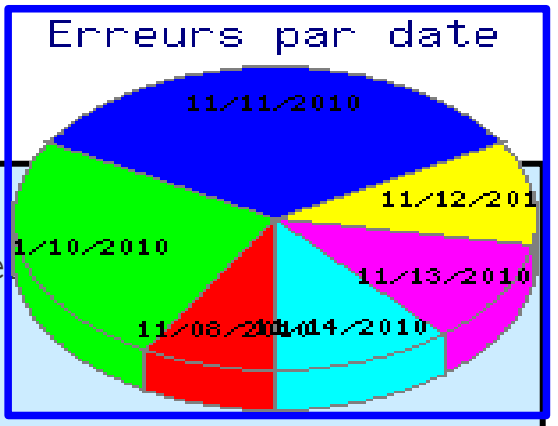
%bydate

11/14/2010	4
11/15/2010	3
11/16/2010	5
11/17/2010	11
11/19/2010	5
11/20/2010	3

@labels	11/14	11/15	11/16	11/17	11/19	11/20
@values	4	3	5	11	5	3

```
for $date (keys(%bydate)) {
    push @labels, $date;
    push @values, $bydate{$date};
}
```

Apache::ParseLog + Graph::Pie



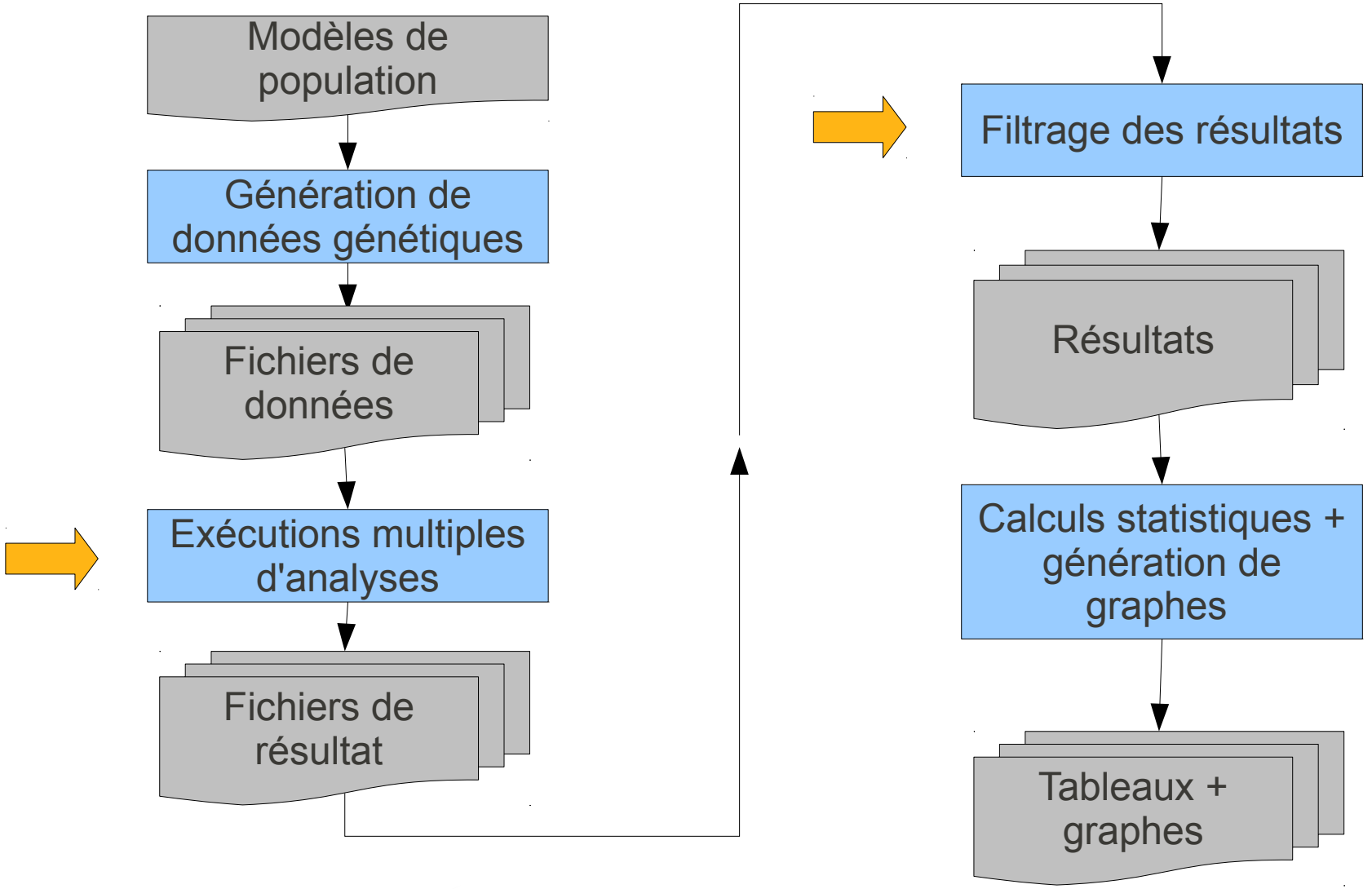

```
use Apache::ParseLog;
$base = new Apache::ParseLog("/etc/apache2/apache
$errorlog = $base->getErrorLog();
%bydate = $errorlog->allbydate();
for $date (keys(%bydate)) {
    push @labels, $date;
    push @values, $bydate{$date};
}
@data = (\@labels, \@values) ;

use GD::Graph;
use GD::Graph::pie;
$graph = GD::Graph::pie->new(600, 400) or die($graph->error);
$graph->set(title => "Erreurs par date") or die($graph->error);
$image = $graph->plot(\@data) or die($graph->error);

 open(F1, '>bydate.png') or die("Cannot write to $file: $!");
 print F1 $image->png ; # image->jpg image->postcript
 close F1;
```

Retour d'expérience

Projet GEPV / Maxime Pauwels
Génétique et Évolution des Populations Végétales

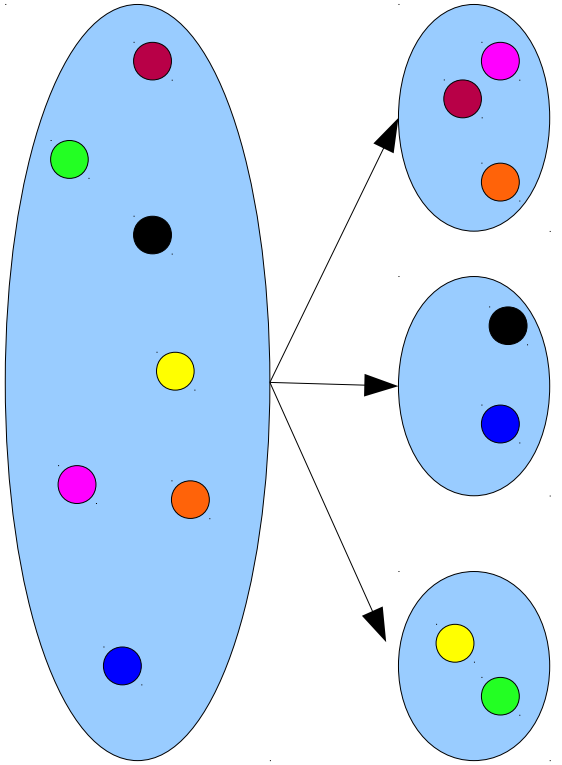
- Enchainement de plusieurs logiciels
- Exécutions répétées d'un des logiciels
  - avec différentes valeurs
- Synthèse des résultats
- Génération de graphiques
- Utilisation du cluster du CRI de l'USTL

Enchaînement de logiciels



Analyses

- Logiciel pour partitionner une population en sous-populations selon la valeur des marqueurs génétiques des individus
  - 1 paramètre K
    - Nombre de sous-populations
    - K varie de 1 à 40
    - Pour chaque valeur de K: 10 réplicats



=> ***400 exécutions (pour un jeu de données)***

Execution en série des analyses

- Logiciel : structure

```
$k_max = 40;
$rep_max = 10;
$resDir = "res";

mkdir $resDir ;
for $k (1 .. $k_max) {
    for $rep (1 .. $rep_max) {
        $out = $resDir . "/k" . $k . "_r" . $rep");
        $cmd = "structure -m mainparams -e extraparams " .
        "-i data -o $out -K $k";
        system($cmd);
    }
}
```

Exemple de nom de fichier résultat res/k6_r4

1 exécution du logiciel

Reprise sur erreur

```
$k_max = 40;
$rep_max = 15;
$resDir = "res";
mkdir $resDir unless (-d $resDir);

for $k (1 .. $k_max) {
    for $rep (1 .. $rep_max) {
        $out = $resDir . "/k" . $k . "_r" . $rep");
        next if (-f $out . "_f");
        $cmd = "structure -m mainparams -e extraparams " .
        "-i data -o $out -K $k";
        system($cmd);
    }
}
```

Si le dossier existe déjà

Si le fichier de résultat existe déjà

Execution en parallèle sur le cluster du CRI Lille1

- Cluster : système de soumission de job
  - PBS : commande qsub
- Contraintes de PBS :
  - Mettre la ligne de commande à exécuter dans un fichier texte (qui sera transmis à qsub)
    - création de 400 fichiers de commande
  - Dans la ligne de commande : utiliser uniquement des chemins complets pour les noms de fichier
    - /home/gallina/bin/mon_programme
    - /home/gallina/projet1/res/k01_r01
  - Ne rien envoyer sur la sortie standard et la sortie d'erreurs

Exécution sur le cluster du CRI Lille1

```
$home = $ENV{"HOME"} ; chomp ($here = `pwd`);
$k_max = 40 ; $rep_max = 10 ; $resDir = "$here/res";
mkdir $resDir unless (-d $resDir);
for $k (1 .. $k_max) {
    for $rep (1 .. $rep_max) {
        $out = $resDir . "/k" . $k . "_r" . $rep");
        next if (-f $out . "_f");
        $cmd = "$home/bin/structure -m $here/mainparams",
            " -e $here/extraparams -i $here/data",
            " -o $out -K $k > /dev/null 2>&1";
        # system($cmd);  version sans utilisation de cluster
        $cmdFile = $out . ".sh";
        open(FHO, ">$cmdFile") or die("cannot create $cmdFile");
        print FHO "$cmd\n";
        close(FHO);
        chmod 755 $cmdFile;
        system("qsub -l nodes=1:ppn=1:xeon $cmdFile");
    }
}
```

Chemins complets

Rien sur les sorties standards

Création du fichier de commandes

Soumission du fichier de commandes

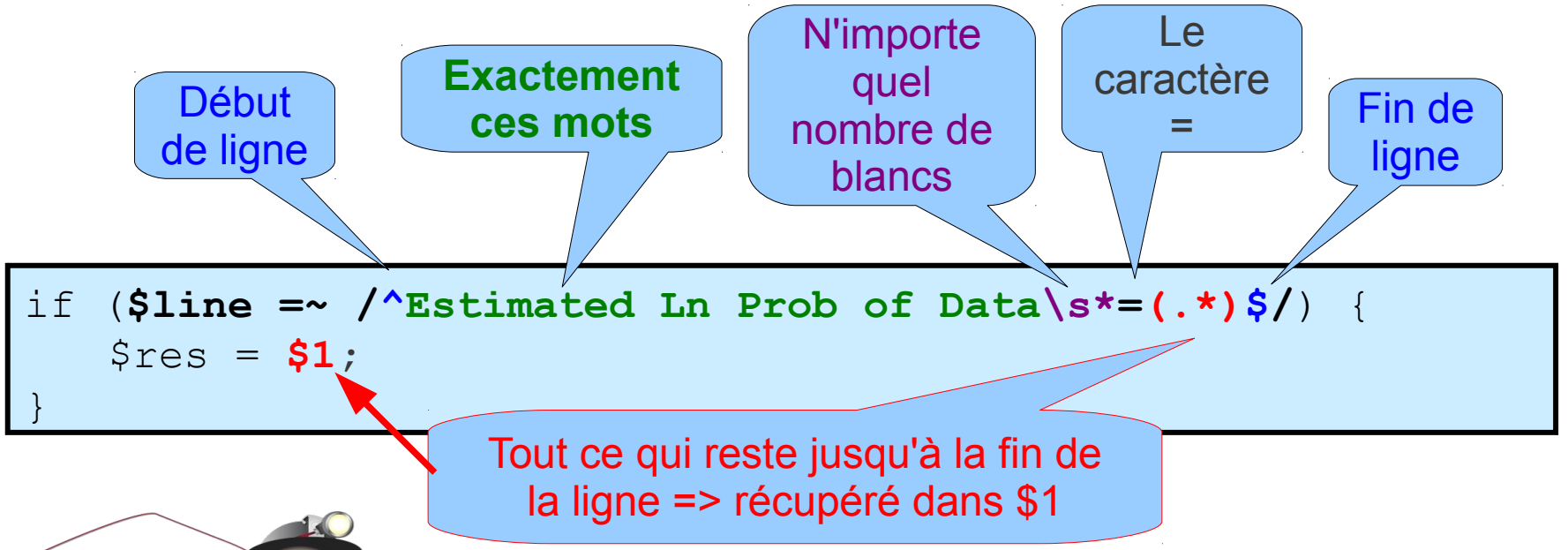
Résultat pour K = 6 et réplicat = 4

```
 2   0.0784     -    0.1022  0.1144  0.0837  0.1356
 3   0.0353  0.1022     -    0.1169  0.0305  0.0314
 4   0.1143  0.1144  0.1169     -    0.1067  0.1248
 5   0.0053  0.0837  0.0305  0.1067     -    0.0584
 6   0.0651  0.1356  0.0314  0.1248  0.0584     -

Average distances (expected heterozygosity) between individuals in same cluster:
cluster  1  : 0.6083
cluster  2  : 0.6091
cluster  3  : 0.6039
cluster  4  : 0.8608
cluster  5  : 0.6230
cluster  6  : 0.5898

---------------------------------------
Estimated Ln Prob of Data   = -22394.5
Mean value of ln likelihood = -21897.8
Variance of ln likelihood   = 993.3
Mean value of alpha         = 0.0348
Allele frequencies uncorrelated


Inferred ancestry of individuals:
        Label (%Miss) Pop:  Inferred clusters
  1   I12-10    (10)   12 :  0.480 0.035 0.034 0.003 0.201 0.248
  2   I12-11     (0)   12 :  0.401 0.004 0.083 0.002 0.500 0.011
  3   I12-12     (0)   12 :  0.805 0.006 0.013 0.002 0.167 0.006
  4   I12-13     (0)   12 :  0.690 0.002 0.017 0.003 0.219 0.069
  5   I12-14     (0)   12 :  0.060 0.036 0.055 0.004 0.038 0.807
  6   I12-15     (0)   12 :  0.355 0.024 0.125 0.007 0.124 0.366
  7   I12-16     (5)   12 :  0.721 0.004 0.009 0.006 0.252 0.008
```

856 lignes …
1 seule m'intéresse

Extraction de données / expressions régulières

- Description (*match*) de la ligne
  - $line **=~** /expression régulière/
  - Retoune vrai si la ligne correspond à l'expression régulière
- Si vrai, on peut récupérer une partie de la chaîne
  - En utilisant des () autour d'une partie de l'expression
- Ligne exemple : `Estimated Ln Prob of Data   = -22394.5`



Collecte des 400 résultats / tableau associatif

- Placer la valeur dans un tableau associatif
  - "*à 2 dimensions*" : K et rep

```
$max_k = 40;
$max_rep = 10;
for $k (1 .. $max_k) {
  for $rep (1 .. $max_rep) {
    $file = $out = $resDir . "/k" . $k . "_r" . $rep") ;
    open(FH, $file) or die("cannot open $file:$!");
    while($line = <FH>) {
      if ($line =~ /^Estimated Ln Prob of Data\s*=(.*)$/){
          $res{$k, $rep} = $1;
          last;
    }
    close(FH);
  }
}
```

Formatage en tableau excel

```
use Spreadsheet::WriteExcel;

$xls = Spreadsheet::WriteExcel->new("res.xls");
$sheet = $xls->add_worksheet("resultats");

# 1ere ligne de titre
$row = 0; $col = 1;
for $k (1 .. $k_max) {
    $sheet->write($row, $col, "k $k"); $col++;
}
$row++; $col = 0;

# lignes de valeurs
for $rep (1 .. $rep_max) {
    $sheet->write($row, $col, "rep $rep"); $col++;
    for $k (1 .. $k_max) {
        $sheet->write($row, $col, $res{$k, $rep}); $col++;
    }
    $row++; $col = 0;
}
```

Créer un doc excel

Créer une feuille

Titres des colonnes

Titres des lignes

Cellules avec les résultats

	k 1	k 2
rep 1	-24747,2	-23053
rep 2	-24747,5	-23051,9
rep 3	-24747,6	-23052,8
rep 4	-24747,5	-23052,3
rep 5	-24747,7	-23049,5
rep 6	-24747,1	-23051,2
rep 7	-24747,6	-23052,9
rep 8	-24747,4	-23051,8
rep 9	-24748	-23052,3
rep 10	-24747,7	-23053,3

Gestion des noms de variables

- Pas besoin de déclarer les variables
- Une variable est créée lors de sa 1ère utilisation

- Pratique pour faire de petits scripts 
- Source d'erreurs pour de + gros développements
  - Fautes de frappe
  - list / liste / lists, k_max / max_k 

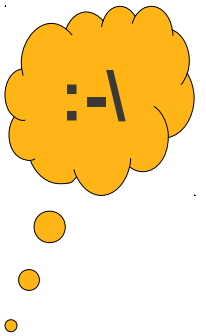
- Directive pour forcer la déclaration des variables
  - use strict;

Déclaration de variables

```
use strict;
sub fonction1 {
   my ($k_max, $rep_max, $resDir, $k, $rep, $out, $cmd);

   $k_max = 40 ; $rep_max = 10;  $resDir = "res";
   for $k (1 .. $max_k) {
      for $rep (1 .. $rep_max) {
         $out = $resDir . "/k" . $k . "_r" . $rep");
         $cmd = "structure -m mainparams -e extraparams " .
         "-i data -o $out -K $k";
         system($cmd);
      }
    }
}
```

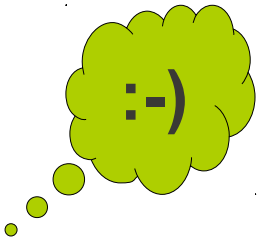
Global symbol "$max_k" requires explicit package name at ./prog1.pl line 6.
Execution of ./prog1.pl aborted due to compilation errors.

Conclusion 1/3
Ce qui rend perplexe au départ: les symboles



- Les noms de variables
  - $ @ %
- Les variables "caractères"
  - @_, $_, $!
- Les expressions régulières
  - /^\s*$/

Conclusion 2/3
Ce qui facilite la vie de programmeur



- Gestion de mémoire automatique
- Tableau associatif
- L'utilisation de tableau sans indices
- Les expressions régulières
- Les packages

Conclusion 3/3
Tâches pour lesquelles perl est adapté

- Traitement de fichiers textes
  - Extraction de données, conversion de format, filtrage
- Automatisation de traitements
  - Enchainement de programmes, exécution multiple
- Génération de synthèses de données
  - Tableaux, graphiques

- Tâches pour lesquelles il existe un package
  - Accès aux bases de données, web
  - Domaines spécifiques : ex biologie BioPerl

Publicité !

- Formation 18-20 janvier 2010

  Introduction à la programmation Perl pour manipuler simplement vos données

- Public : personnes qui doivent traiter de grandes quantités de données (biologistes, chimistes etc...)
- Intervenants : Stefan Gaget & Sophie Gallina
- Date limite d'inscription : 10 décembre 2010

Questions ?