LANGAGES DE SCRIPT
LES ALTERNATIVES AUJOURD'HUI

Jacquelin Charbonnel

Journées Mathrice – Dijon – Mars 2011

Langages de script

- à l'origine
  - langage de macro-commandes
  - *huile* inter application

```
#!/bin/bash

mkdir /users/alfred
usermod -d /users/alfred alfred
passwd alfred
groupadd theproject
usermod -G theproject alfred
```

Langages de script

- à l'origine
  - langage de macro-commandes
  - *huile* inter application
- + variables + arguments

```
#!/bin/bash

login=$1
group=$2

mkdir /users/$login
usermod -d /users/$login $login
passwd $login
groupadd $group
usermod -G $group $login
```

Langages de script

- à l'origine
  - langage de macro-commandes
  - *huile* inter application
- + variables + arguments
- + des commandes internes

```
read r
```

Langages de script

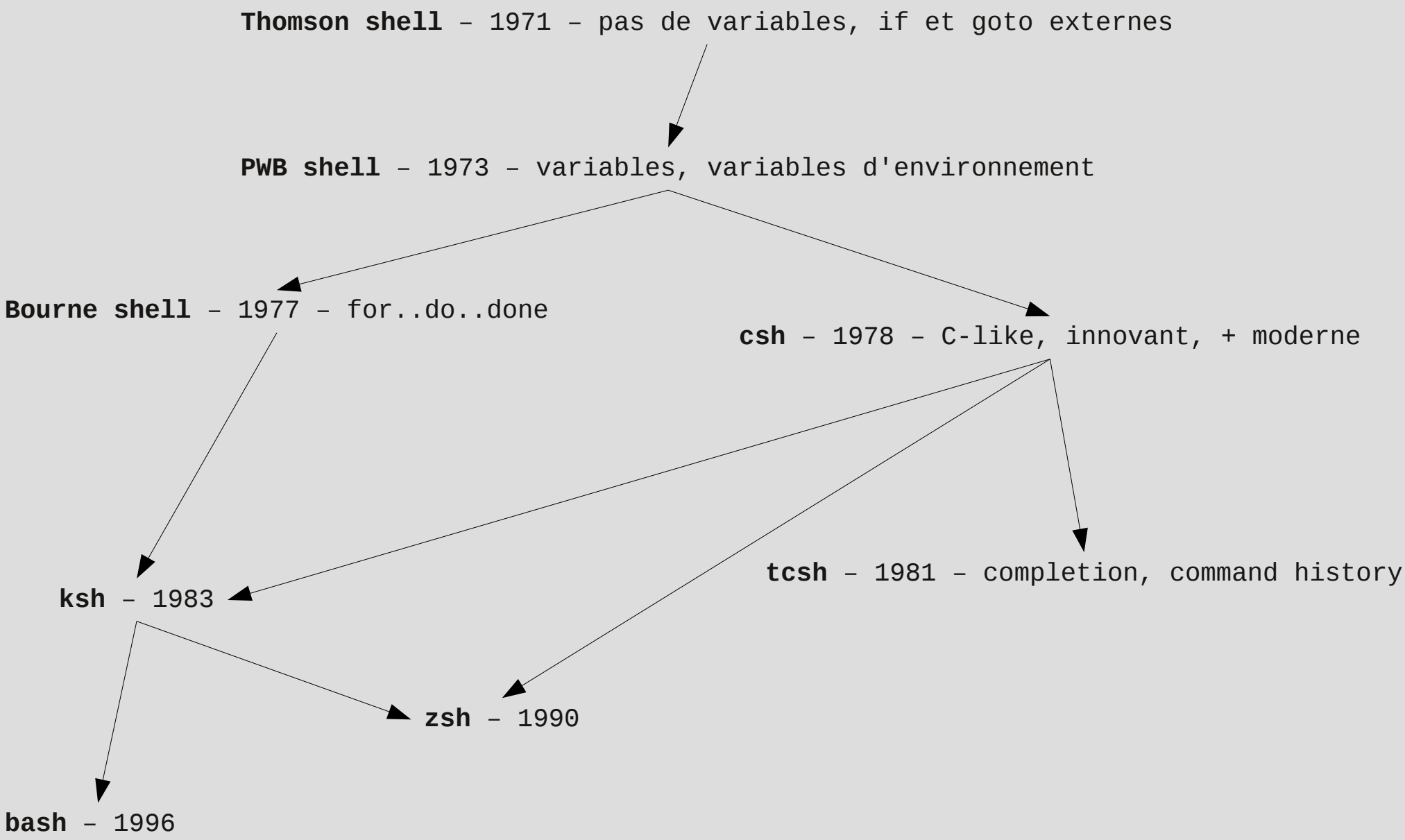
- à l'origine
  - langage de macro-commandes
  - *huile* inter application
- + variables + arguments
- + des commandes internes
- + des conditions

```
if ! echo "$r"|grep '^[yYoO]' ; then echo "aborted !" ; exit 1 ; fi
```

Langages de script

- à l'origine
  - langage de macro-commandes
  - *huile* inter application
- + variables + arguments
- + des commandes internes
- + des conditions
- + des boucles

Petite histoire des shells Unix



- Ken Thompson :
  - l'homme qui créa le langage B !

```
if ... fi
case ... esac
```

- Ken Thompson :
  - l'homme qui créa le langage B !

```
if ... fi
case ... esac
```

```
for ... do ... done
while ... do ... done
until ... do ... done
```

Bourne branche

```
foreach ... end
while ... end
```

csh branche

- Ken Thompson :
  - l'homme qui créa le langage B !

```
if ... fi
case ... esac
```

```
for ... do ... done
while ... do ... done
until ... do ... done
```

Bourne branche

```
foreach ... end
while ... end
```

csh branche

```
for ... rof
until ... litnu
while ... elihw
```

Exercice

- Y voir un peu plus clair dans les logs de sendmail | postfix

```
Jul 25 04:02:09 tonton postfix/local[5432]: 52BD11744C:
to=<mailmaster@tonton.univ-angers.fr>, orig_to=<root>, relay=local, delay=6.8,
delays=3.5/0/0/3.3, dsn=2.0.0, status=sent (delivered to command:
/usr/bin/procmail)
Jul 25 04:02:09 tonton postfix/qmgr[2125]: 52BD11744C: removed
Jul 25 04:03:05 tonton postfix/smtpd[5426]: disconnect from
netsrv.math[172.19.45.20]
Jul 25 04:03:05 tonton postfix/qmgr[2125]: 479521744B: from=<root@netsrv.math>,
size=4418700, nrcpt=1 (queue active)
Jul 25 04:03:05 tonton spamc[6085]: skipped message, greater than max message size
(512000 bytes)
Jul 25 04:03:05 tonton postfix/local[5427]: 479521744B: to=<mailmaster@smtp.math>,
relay=local, delay=60, delays=59/0/0/0.36, dsn=2.0.0, status=sent (delivered to
command: /usr/bin/procmail)
```

- ne garder que les lignes contenant to= ou from=

```
$ grep -E ' to=| from=' /var/log/maillog

Jul 25 10:43:02 tonton postfix/qmgr[2125]: F17EB1744A: from=<root@tonton.univ-
gers.fr>, size=3140, nrcpt=1 (queue active)
Jul 25 10:43:02 tonton postfix/local[21913]: F17EB1744A: to=<mailmaster@smtp.m  h>,
relay=local, delay=0.88, delays=0.05/0/0/0.83, dsn=2.0.0, status=sent (del  ered to
command: /usr/bin/procmail)
Jul 25 10:45:36 tonton postfix/qmgr[2125]: 2553F1744A: from=<winyourbuyin-3960
vica-alpha.com>, size=11285, nrcpt=1 (queue active)
Jul 25 10:45:39 tonton postfix/local[22556]: 2553F1744A: to=<jea@tonton.univ-a
ers.fr>, relay=local, delay=3.3, delays=0.04/0.01/0/3.2, dsn=2.0.0, status=sen
(delivered to command: /usr/bin/procmail)
```

- ne garder que l'info qui nous intéresse

```
$ grep -E ' to=| from=' /var/log/maillog | awk '{print $6,$7}' | sort

1B9B11744A: to=<mailmaster@smtp.math>,
BE8D91744A: from=<famille_loustau-bounces@listes.clubdefrance.com>,
BE8D91744A: to=<loustau@math.univ-angers.fr>,
50A1A1744A: from=<root@ns1.netmize.org>,
50A1A1744A: to=<landreau@math.univ-angers.fr>,
4D42A1744A: from=<VGIRARDI@aol.com>,
4D42A1744A: to=<granger@tonton.univ-angers.fr>,
96DC01744A: from=<mikec7@me.com>,
96DC01744A: to=<schaub@math.univ-angers.fr>,
```

- 1 seule ligne par mail

```
$ grep -E ' to=| from=' maillog |awk '{print $6,$7}'| sort | sed '/^.*$/N;s/\n/ /'

008181744A: from=<root@tonton.univ-angers.fr>, 008181744A: to=<mailmaster@smtp.math>,
0087C1744A: from=<email@newsletter.pgr9.com>, 0087C1744A: to=<schaub@math.univ-
angers.fr>,
008F817453: from=<Philippe.Depouilly@math.u-bordeaux1.fr>, 008F817453:
to=<jaclin@math.univ-angers.fr>,
009801744C: from=<sg@snesup.fr>, 009801744C: to=<lucas@tonton.univ-angers.fr>,
```

- ne garder que les champs intéressants

```
$ grep -E ' to=| from=' maillog |awk '{print $6,$7}'| sort | sed '/^.*$/N;s/\n/ /' |
awk '{print $1,$2,$4}'

008181744A: from=<root@tonton.univ-angers.fr>, to=<mailmaster@smtp.math>,
0087C1744A: from=<email@newsletter.pgr9.com>, to=<schaub@math.univ-angers.fr>,
008F817453: from=<Philippe.Depouilly@math.u-bordeaux1.fr>, to=<jaclin@math.univ-
angers.fr>,
009801744C: from=<sg@snesup.fr>, to=<lucas@tonton.univ-angers.fr>,
```

bash script

- + des fonctions

```
usage()
{
  echo "Usage: $0 username dbname"
  exit 2
}
```

bash script

- + des fonctions
- + de l'arithmétique

```
a=3
b=2

c=`expr \( $a + $b \) / 2`    # pour faire (3+2)/2 !
d=$(( ( $a + $b ) / 2 ))      # idem en + moderne
```

bash script

- + des fonctions
- + de l'arithmétique
- + des tableaux

```
#!/bin/bash

declare -a mon_tableau   # déclare le tableau

mon_tableau=( nuage soleil neige )

echo ${mon_tableau[2]}

echo ${mon_tableau[*]}   # tous les éléments

nb=${#mon_tableau[*]}
echo $nb                 # le nombre d'élément

for ((i=0;i<$nb;i++)); do
  echo ${mon_tableau[${i}]}
done
```

STOP !

Pourquoi ne pas utiliser un vrai langage de programmation ?!?

- De quoi l'ASR a-t-il besoin ?
  - développer rapidement de petits scripts utilitaires
  - jongler en désinvolte avec les chaînes de caractères
  - en option :
    - programmer des algorithmes complexes
    - profiter de structures de données évoluées
  - efficacité, performance ? bof !

- Ce qu'il faut :
  - faiblement typé
    - erreurs détectées le + tard possible (voire pas du tout ! ☺)
  - syntaxe concise (on n'a pas que ça à faire !)
  - traitement agréable des re
  - des tableaux, des listes, des hashs

- awk (1977)
- REXX (1982)
- Perl (1987)
- TCL (1988)
- Python (1991)
- LUA (1994)
- Ruby (1995)
- PHP (1995)
- Pike (1996)
- Scriptol (2001)
- Go (2009)

- awk (1977)
- REXX (1982)
- Perl (1987)
- TCL (1988)
- Python (1991)
- LUA (1994)
- Ruby (1995)
- PHP (1995)
- Pike (1996)
- Scriptol (2001)
- Go (2009)

Concision : le minimum

- le plus petit programme qui ne fait rien

	bash	awk	Perl	Python	Ruby	C	Java
# car.	0	0	0	0	0	20	68

```
#include <stdio.h>
main() { return 0 ; }
```

```
public class smallest { public static void main(String[] args) { }
```

Concision : hello world

	bash	awk	Perl	Python	Ruby	C	Java
car.	16	26	21	19	18	69	115

```
echo Hello World
```

```
BEGIN{print "Hello Wolrd"}
```

```
print "Hello World\n"
```

```
print "Hello World"
```

```
puts "Hello World"
```

```
#include <stdio.h>
main() {
    puts("Hello World");
    return 0;
}
```

```
public class hello_world {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

awk

- A. Aho, P. Weinberger et B. Kernighan (Bell labs)
- filtre


```
#!/usr/bin/awk

pattern { instr }
pattern { instr }
pattern { instr }
```

```
#!/usr/bin/awk

BEGIN { inst }
END { instr }
pattern { instr }
pattern { instr }
pattern { instr }
```

awk

```
$ awk '/root/' /etc/passwd

root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

```
$ awk -F : '/root/ {print $1,$3}' /etc/passwd

root 0
operator 11
```

```
$ ll |  awk 'BEGIN {s=0}
                 {s=s+$5 ; print}
            END   {print "total size:",s}'

-rwxr-xr-x 1 jaclin users  73 Aug 17 11:13 1.py
-rwxr-xr-x 1 jaclin users 211 Aug 17 17:08 1.rb
-rwxr-xr-x 1 jaclin users 366 Aug 17 19:17 1.sh
-rwxr-xr-x 1 jaclin users  81 Aug 16 08:16 1.tcl
total size:  731
```

Perl

- *Practical Extraction and Report Language*
- Larry Wall (*seul informaticien vivant ayant une rue à son nom*)
  - *"There Is More Than One Way To Do It"*
- objectif à l'époque :
  - extraire des données et sortir des rapports bien formatés
  - super awk

```
perl -n -e '/root/ && print' /etc/passwd

root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

```
perl -anF: -e '/root/ && print "$F[0] $F[2]\n"' /etc/passwd

root 0
operator 11
```

Perl

- *Practical Extraction and Report Language*
- Larry Wall (*seul informaticien vivant ayant une rue à son nom*)
  - *"There Is More Than One Way To Do It"*
- objectif à l'époque :
  - extraire des données et sortir des rapports bien formatés
  - super awk

awk / perl

```
ll | perl -ane ' BEGIN {$s=0}
                 $s+=$F[4]; print ;
                 END { print "total size: $s\n"}'

-rwxr-xr-x 1 jaclin users  73 Aug 17 11:13 1.py
-rwxr-xr-x 1 jaclin users 211 Aug 17 17:08 1.rb
-rwxr-xr-x 1 jaclin users 366 Aug 17 19:17 1.sh
-rwxr-xr-x 1 jaclin users  81 Aug 16 08:16 1.tcl
total size: 731
```

```
$ ll |  awk 'BEGIN {s=0}
                 {s=s+$5 ; print}
            END   {print "total size: ",s}'

-rwxr-xr-x 1 jaclin users  73 Aug 17 11:13 1.py
-rwxr-xr-x 1 jaclin users 211 Aug 17 17:08 1.rb
-rwxr-xr-x 1 jaclin users 366 Aug 17 19:17 1.sh
-rwxr-xr-x 1 jaclin users  81 Aug 16 08:16 1.tcl
total size:  731
```

filtres en Perl

```
#!/usr/bin/perl

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

```
$ cat /etc/passwd | ./myprog
$ ./myprog /etc/passwd
$ ./myprog /var/log/*.log
```

Perl doué pour traiter les chaînes

```
#!/usr/bin/perl

while (<>)
{
  s/#.*// ;
  print ;
}
```

data

```
# exemple de fichier do données

un = 1            # premiere valeur
  deux=  2
trois = un+deux
```

```
un = 1
  deux=  2
trois = un+deux
```

Perl doué pour traiter les chaînes

```
#!/usr/bin/perl

while (<>)
{
  s/#.*// ;
  next if /^\s*$/ ;
  print ;
}
```

data

```
# exemple de fichier do données

un = 1           # premiere valeur
  deux=  2
trois = un+deux
```

```
un = 1
  deux=  2
trois = un+deux
```

Perl doué pour traiter les chaînes

```
#!/usr/bin/perl

while (<>)
{
  chomp ;
  s/#.*// ;
  next if /^\s*$/ ;
  ($var,$val) = /           =           / ;
}
```

```
# exemple de fichier
un = 1           # premiere valeur
  deux=  2
trois = un+deux
```

Perl doué pour traiter les chaînes

```
#!/usr/bin/perl

while (<>)
{
  chomp ;
  s/#.*// ;
  next if /^\s*$/ ;
  ($var,$val) = /^   (\S+)   =   (.*)   $/ ;
}
```

```
# exemple de fichier
un = 1           # premiere valeur
  deux=  2
trois = un+deux
```

Perl doué pour traiter les chaînes

```
#!/usr/bin/perl

while (<>)
{
  chomp ;
  s/#.*// ;
  next if /^\s*$/ ;
  ($var,$val) = /^\s*(\S+)\s*=\s*(.*)\s*$/ ;
}
```

```
# exemple de fichier
un = 1           # premiere valeur
  deux=  2
trois = un+deux
```

Perl doué pour traiter les chaînes

```
#!/usr/bin/perl

while (<>)
{
  chomp ;
  s/#.*// ;
  next if /^\s*$/ ;
  ($var,$val) = /^\s*(\S+)\s*=\s*(.*)\s*$/ ;
  $h->{$var} = $val ;
}
```

```
# exemple de fichier
un = 1           # premiere valeur
  deux=  2
trois = un+deux
```

```
 {
          'trois' => 'un+deux',
          'un' => '1',
          'deux' => '2'
 }
```

Les re de Perl

- Perl étend les re de sed, awk, egrep, vi...
- les re de Perl ont connu un grand succès
  - bibliothèque PCRE (Perl Compatible Regular Expressions)
  - sont maintenant incluses dans de nombreux langages
- ont donné à Perl une réputation de langage peu lisible

programmes Perl illisibles ?

```
#!/usr/bin/perl

while (<>)
{
  chomp ;
  s/\s*#.*$// ;
  next if /^\s*$/ ;
  ($var,$val) = /^\s*(\S+)\s*=\s*(.*)\s*$/ ;
  $h->{$var} = $val ;
}
```

programmes Perl illisibles ?

```
#!/usr/bin/perl

while (<>)
{
  chomp ;
  s/      // ;
  next if /     / ;
  ($var,$val) = /                       / ;
  $h->{$var} = $val ;
}
```

Les listes de Perl

```
@l1 = ("un","deux","trois") ;
for $i (@l1) { print "$i " } # -> un deux trois

$l1[1] = "quatre" ;           # -> un quatre trois

pop @l1 ;                     # -> un quatre
push @l1,('trois',"cinq") ;   # -> un quatre trois cinq

shift @l1 ;                   # -> quatre trois cinq
unshift @l1,"six" ;           # -> six quatre trois cinq

($a,$b,@l3) = @l1 ;           # -> $a=six, $b=quatre, @l3=(trois,cinq)

@l2 = ( "a",@l1,"b" ) ;       # -> a six quatre trois cinq b
```

Arguments de fonction en Perl

```
sub f
{
  ($a,$b) = @_ ;
  ...
}

f(1.2,"tagada") ;
```

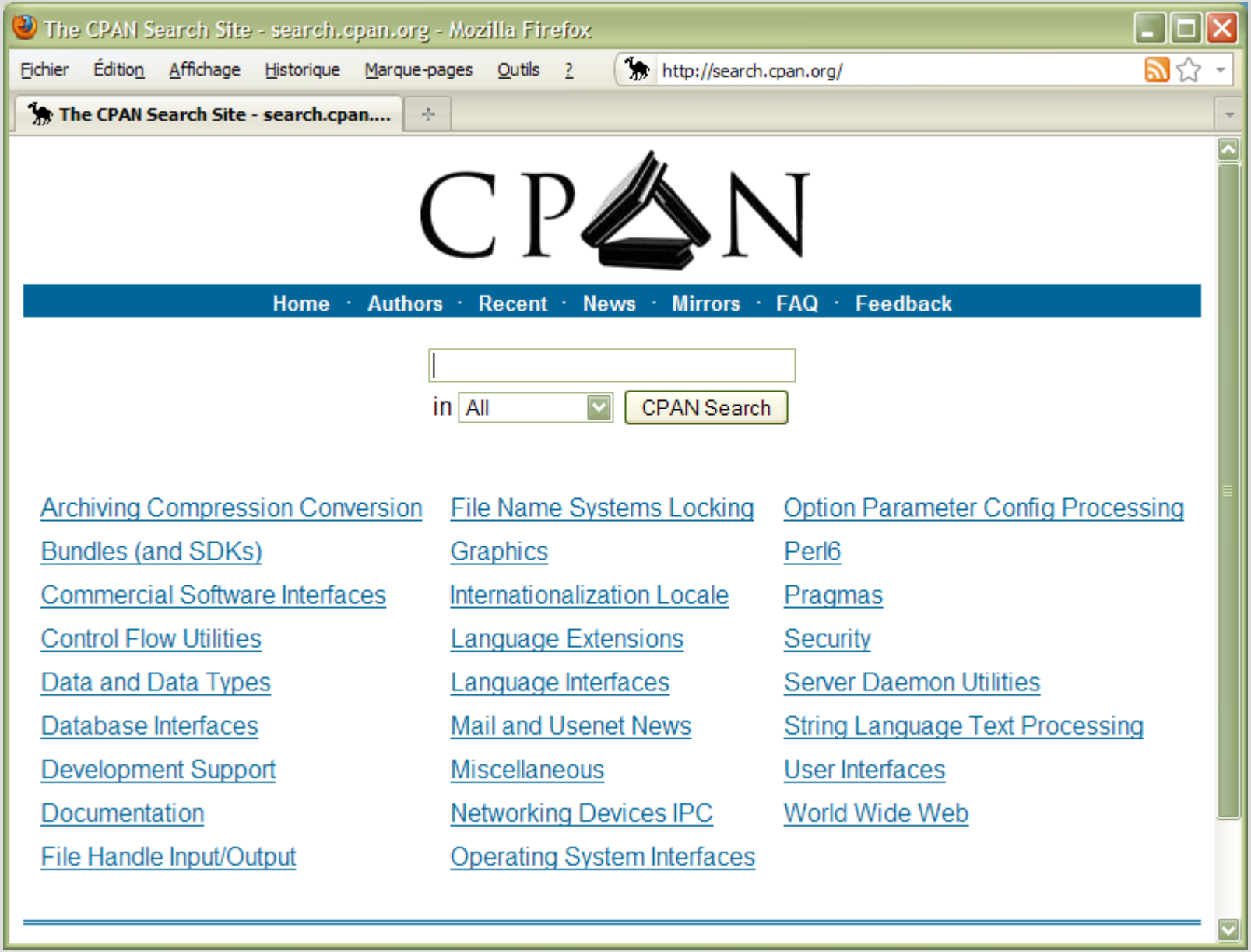
Arguments de fonction en Perl

```
sub f
{
  my (@p1,@p2) = @_ ;

  printf("\@p1: %s\n",join(",",@p1)) ;
  printf("\@p2: %s\n",join(",",@p2)) ;
}

@l1 = (1,2,3) ;
@l2 = (4,5) ;
f(@l1,@l2) ;
```

```
@p1: 1,2,3,4,5
@p2:
```

Les hashes de Perl

```
%h = ( un => 1 , deux => 2 , trois => 3 ) ;
# { 'trois' => 3, 'un' => 1, 'deux' => 2 }

$h{quatre} = 4 ;
# { 'trois' => 3, 'un' => 1, 'quatre' => 4, 'deux' => 2 }

delete $h{deux} ;
# { 'trois' => 3, 'un' => 1, 'quatre' => 4 }

@k = keys %h ;
# ( 'trois', 'un', 'quatre' }

@v = values %h ;
# { 3, 1, 4 }

for $k (keys %h) { print "($k->$h{$k}) " ; }
# (first->f) (un->1) (trois->3) (last->l) (quatre->4)
```


The CPAN Search Site - search.cpan.org - Mozilla Firefox
Fichier Édition Affichage Historique Marque-pages Outils ?
http://search.cpan.org/
The CPAN Search Site - search.cpan....
CPAN
Home · Authors · Recent · News · Mirrors · FAQ · Feedback
in All
CPAN Search
Archiving Compression Conversion
Bundles (and SDKs)
Commercial Software Interfaces
Control Flow Utilities
Data and Data Types
Database Interfaces
Development Support
Documentation
File Handle Input/Output
File Name Systems Locking
Graphics
Internationalization Locale
Language Extensions
Language Interfaces
Mail and Usenet News
Miscellaneous
Networking Devices IPC
Operating System Interfaces
Option Parameter Config Processing
Perl6
Pragmas
Security
Server Daemon Utilities
String Language Text Processing
User Interfaces
World Wide Web

Le même exercice en Perl

```
#!/usr/bin/perl

@ARGV = ("/var/log/maillog") ;

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

```
Jul 25 04:02:09 tonton postfix/local[5432]: 52BD11744C: to=<mailmaster@tonton.univ-
angers.fr>, orig_to=<root>, relay=local, delay=6.8, delays=3.5/0/0/3.3, dsn=2.0.0,
status=sent (delivered to command: /usr/bin/procmail)
Jul 25 04:02:09 tonton postfix/qmgr[2125]: 52BD11744C: removed
Jul 25 04:03:05 tonton postfix/smtpd[5426]: disconnect from netsrv.math[172.19.45.20]
Jul 25 04:03:05 tonton postfix/qmgr[2125]: 479521744B: from=<root@netsrv.math>,
size=4418700, nrcpt=1 (queue active)
```

Le même exercice

```
#!/usr/bin/perl

@ARGV = ("/var/log/maillog") ;

while(<>)
{
  ($id,$rest) = /([0-9A-F]{10}):(.*)/ ;
  print if $id ;
}
```

```
Jul 25 04:02:09 tonton postfix/local[5432]: 52BD11744C: to=<mailmaster@tonton.univ-
angers.fr>, orig_to=<root>, relay=local, delay=6.8, delays=3.5/0/0/3.3, dsn=2.0.0,
status=sent (delivered to command: /usr/bin/procmail)
Jul 25 04:02:09 tonton postfix/qmgr[2125]: 52BD11744C: removed
Jul 25 04:03:05 tonton postfix/qmgr[2125]: 479521744B: from=<root@netsrv.math>,
size=4418700, nrcpt=1 (queue active)
```

Le même exercice

```
#!/usr/bin/perl

@ARGV = ("/var/log/maillog") ;

while(<>)
{
  ($id,$rest) = /([0-9A-F]{10}):(.*)/ ;
  push(@{$h->{$id}},$rest) if $id ;
}
```

```
'E4A9F1744B' => [
  ' client=math.univ-angers.fr[193.49.146.25]',
  ' message-id=<4C51974F.9090401@certa.ssi.gouv.fr>',
  ' from=<corresp_ssi-owner@services.cnrs.fr>, size=13207, nrcpt=1',
  ' to=<jaclin@math.univ-angers.fr>, status=sent',
  ' removed'
  ],
'916561744B' => [
  ' client=math.univ-angers.fr[193.49.146.25]',
  ' message-id=<OP-M80-S55-1280040877@rem02.com>',
  ' from=<bounce@rem02.com>, size=6585, nrcpt=1',
  ' to=<jea@tonton.univ-angers.fr>, status=sent',
  ' removed'
],
```

Quelques particularités de Perl

- sensibilité au contexte

```
#!/usr/bin/perl

open(F,"/etc/passwd") or die ;
$contenu = <F> ;     # 1 ligne
@lignes = <F> ;      # le reste des lignes
```

Autres particularités de Perl

- heredoc : 3 formes

```
#!/usr/bin/perl

$var="deux" ;
$msg = << "EOL" ;
un
$var
trois
EOL

print $msg ;
```

```
un
deux
trois
```

```
#!/usr/bin/perl

$var="deux" ;
$msg = << 'EOL' ;
un
$var
trois
EOL

print $msg ;
```

```
un
$var
trois
```

```
#!/usr/bin/perl

$var="deux" ;
print << `EOL` ;
pwd
ls
EOL
```

Autres particularités de Perl

- use strict : fini le laxisme !

```
#!/usr/bin/perl

use strict ;

$var="blabla" ;
print $var ;
```

```
Global symbol "$var" requires explicit package name at str line 5.
Global symbol "$var" requires explicit package name at str line 6.
Execution of str aborted due to compilation errors.
```

```
#!/usr/bin/perl

use strict ;

my $var="blabla" ;
print $var ;
```

Autres particularités de Perl

- Perl s'en sort toujours

```
$a = "1" ;
$b = 2 ;

print $a+$b,"\n" ;
print $a.$b,"\n" ;
```

```
3
12
```

Perl en résumé

- syntaxe concise
- manipulation natives des re
- langage unique pour maquetter et finaliser :
  - maquettage très rapide
  - puis quand ça ne va plus (script trop gros) :
    - use strict ;
  - et on finalise
- richesse des modules CPAN

- CMS : Mason, bricolage
- Wiki : Twiki, MojoMojo
- Sympa, SpamAssassin, AWstats, MRTG, Urpmi, Webmin, Bugzilla, SVK, Fink, automake

- Perl permet de faire des choses très vite
  - (et accessoirement aussi très mal)

Python

- Guido Van Rossum + Monthy Python
- à la mode : Google, MicroSoft, RedHat
- syntaxe simple, épurée, lisible...

```
#!/usr/bin/python
import re

f = open('/var/log/maillog', 'r')
dict = {}
for line in f.readlines() :
        pattern = re.search(r"([0-9A-F]{10}):(.*)",line)
        if pattern:
                if not pattern.group(1) in dict :
                        dict[pattern.group(1)] = []
                dict[pattern.group(1)].append(pattern.group(2))
```

le même exercice en Python

... à condition de bannir le retour automatique à la ligne, d'avoir des fenêtres et imprimantes larges !

```
#!/usr/bin/python
import re

f = open('/var/log/maillog', 'r')
dict = {}
for line in f.readlines() :
        pattern = re.search(r"([0-9A-
F]{10}):(.*)",line)
        if pattern:
                if not
pattern.group(1) in dict :

dict[pattern.group(1)] = []

dict[pattern.group(1)].append(pattern.
group(2))
```

```
$ file ./awn/awn-settings/awnClass.py
./awn/awn-settings/awnClass.py: ASCII Pascal program text
```

```
$ file ./awn/awn-settings/awnClass.py
./awn/awn-settings/awnClass.py: ASCII Pascal program text
```

```
import sys
import os
import socket
import time
import urllib
import cairo
import array
from ConfigParser import ConfigParser
try:
    from cStringIO import StringIO
except ImportError:
    from StringIO import StringIO
try:
    import gobject
    import gtk
    import gtk.gdk as gdk
    import pango
except Exception, e:
    sys.stderr.write(str(e) + '\n')
    sys.exit(1)
from xdg.DesktopEntry import DesktopEntry
```

```
$ file ./awn/awn-settings/awnClass.py
./awn/awn-settings/awnClass.py: ASCII Pascal program text

$ file ./menumaker-0.99.7/Config.py
./menumaker-0.99.7/Config.py: ASCII text

$ file ./menumaker-0.99.7/MenuMaker/WindowMaker.py
./menumaker-0.99.7/MenuMaker/WindowMaker.py: ASCII Java program text

$ file ./awn/tests/test-awn-dialog.py
./awn/tests/test-awn-dialog.py: a python script text executable
```

- POO très bien supportée
- % Perl, on perd l'utilisation naturelle des re

```
import re
m = re.search('(?<=abc)def', 'abcdef')
```

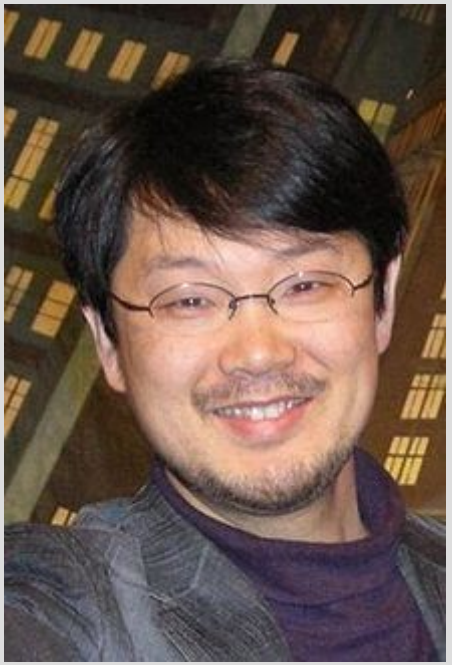
Python aujourd'hui

- Bibliothèques de calcul :
  - Calcul : NumPy, SciPy, PyIMSL Studio, Sympy, SAGE
- CMS
  - Plone, Zope, MoinMoin
- Google App Engine
- yum, anaconda, kvm, SAGE

- ironie du sort
  - l'absence de ponctuation délimitant les blocs est aujourd'hui un obstacle (cf http://lambda-the-ultimate.org/node/1480)

- Python OBLIGE une certaine structure donc
  - c'est plus long de faire des choses simples
  - mais elles restent plus facile à lire 6 mois après

Ruby



- Yukihiro Matsumoto
  - *éviter à l'utilisateur (de façon la plus simple possible) les mauvaises surprises*
- clin d'oeil à Perl pour le nom
- proche de Perl
- tout est objet ; les classes sont dynamiques
  - possibilité de modifier les classes (prédéfinies ou non) à l'exécution

Syntaxe originale

- usage intensive d'itérateurs :

```
5.times { print "Ohe !" }

5.times { |i| print i," " }
```

```
File.open("/etc/passwd", "r").each_line { |ligne| puts ligne }
```

```
somme = 0
tab.each { |i| somme += i }
print somme
```

Syntaxe originale

- concise mais néanmoins lisible

```
#!/usr/bin/ruby

h = {}
File.open('/var/log/maillog','r+').each_line { |e|
   next unless e =~ /([0-9A-F]{10}):(.*)/
   data = Regexp.last_match
   if h[data[1]]==nil then
        h[data[1]] = []
   end
   h[data[1]] << data[2]
}
```

le même exercice en Ruby

Syntaxe originale

- ponctuation pour faciliter la lisibilité
  - les fonctions booléennes se terminent par ?
  - le ! signifie que la fonction modifie la valeur de l'objet

```
puts "#{i} est un entier" if i.integer?
```

```
a = [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
a.sort!
```

Ruby proche de Perl

- re natives

```
johnsays = “It's 9:18PM here now.I cannot wait to go out tonight.”
if johnsays =~ /\d:\d\d/
    puts “John told you what time it is.”
else
    puts “John does not care what time it is.”
end
```

fonction classique

```
def demo
   puts "Et voici le code bloc :"
   puts "C'était le code bloc !"
end

demo
```

```
Et voici le code bloc :
C'était le code bloc !
```

code bloc

```
def demo
   puts "Et voici le code bloc :"
   yield
   puts "C'était le code bloc !"
end

demo {puts "### je suis le code bloc ###"}
```

```
Et voici le code bloc :
### je suis le code bloc ###
C'était le code bloc !
```

arguments de code bloc

```
def demo
   puts "Voici 2 nombres aléatoires : "
   yield rand(10), rand(50)
   puts "On dit merci !"
end

demo {|x,y| puts "#{x}, #{y}" }
demo {|x,y| print "*"*x,"-"*y,"\n"}
```

```
Voici 2 nombres aléatoires :
9, 33
On dit merci !

Voici 2 nombres aléatoires :
***-------------------------------------------
On dit merci !
```

arguments de code bloc

```
def demo(max)
   puts "Voici 2 nombres aléatoires (entre 0 et #{max}) :"
   yield rand(max), rand(max)
   puts "On dit merci !"
end

demo(100) {|x,y| puts "#{x}, #{y}" }
demo(10) {|x,y| print "*"*x,"-"*y,"\n"}
```

```
Voici 2 nombres aléatoires (entre 0 et 100) :
75, 83
On dit merci !

Voici 2 nombres aléatoires (entre 0 et 10) :
**----
On dit merci !
```

Itérateur

- 3 itérateurs prédéfinis
  - each, find, collect

```
[ 1, 3, 5 ].each { |i| puts i }

[1, 3, 5, 7, 9].find {|v| v*v > 30 }  » 7

["H", "A", "L"].collect { |x| x.succ }  » ["I", "B", "M"]
```

```
File.open("testfile").each do { |line| print line }
```

Ruby aujourd'hui

- RRA (Ruby Application Archive), RubyForge
- rubygem

- Ruby on Rail
- Puppet
- Redmine

Où en est-on en 2011 ?

- Perl 6
  - nouveau langage, incompatible avec perl 5.x
- Python 3
  - en partie incompatible avec python 2.x
- Ruby

Pourquoi Perl 6 ?

- au départ, Perl est un langage très modeste :
  - conçu pour des programmes de quelques lignes (super awk)
- a grossi en respectant la compatibilité ascendante
- objectif de Perl 6 :
  - toiletter & homogénéiser la syntaxe
  - intégrer de nouvelles fonctionnalités
  - mieux exploiter les architectures matérielles modernes

Le développement de Perl 6

- officialisation par Larry Wall en 2000, qui annonce l'objectif :
  - *“les choses faciles doivent rester faciles, les choses difficiles doivent être plus faciles, et les choses impossibles ne devraient pas être difficiles”*
- appel à idées public
- 2006
  - Pugs, première mise en œuvre expérimentale (écrite en Haskell)

Perl 6

- refonte totale du langage
- garde le même esprit :
  - laisser une grande liberté au programmeur
  - syntaxe concise : principe de Huffman
- complètement incompatible avec Perl 5
  - module de compatibilité prévu
- soin tout particulier apporté à la syntaxe du langage :
  - lisibilité
  - extensibilité

Principe de Huffman

- appliqué à la syntaxe d'un langage de programmation
  - l'encodage d'un symbole souvent utilisé doit être plus court que celui d'un symbole moins utilisé
- objectif : obtenir des programmes concis
- pourquoi ?
  - pour voir plus de code sur son écran
    - voir + facilement des similarités /différences
    - éviter de faire défiler les lignes sur son écran

Nouveautés de Perl 6

- passage d'arguments revu
- modèle objet revu
- superposition
  - analogie avec la superposition quantique
    - occuper plusieurs états jusqu'à ce que l'observation les effondre

```
my $un_chiffre = 0|2|4|6|8;        # any(0, 2, 4, 6, 8)
my $cinq_chiffres = 1&3&5&7&9;     # all(1, 3, 5, 7, 9)

$un_chiffre += 1;                  # maintenant égal à 1|3|5|7|9
$un_chiffre += (1&2)               # (1|3|5|7|9)&(2|4|6|8|10)
```

Nouveautés de Perl 6

- système d'expressions régulières entièrement refondu
  - possibilité d'écrire des grammaires

```
grammar URL {
    token TOP {
        <schema> '://' [<hostname> | <ip> ] [ ':' <port>]? '/' <path>?
    }
    token byte { (\d**{1..3}) <?{ $0 < 256 }> }
    token ip { <byte> [\. <byte> ] ** 3 }
    token schema { \w+ }
    token hostname { (\w+) ( \. \w+ )* }
    token port { \d+ }
    token path { <[ a..z A..Z 0..9 -_.!~*'():@&=+$,/ ]>+ }
}

my $match = URL.parse('http://perl6.org/documentation/');
say $match<hostname>   # affichera "perl6.org"
```

Perl 6 en 2011

- version non stabilisée
  - tourne sur une machine virtuelle (Parrot)
  - le compilateur s'appelle rakudo (« la voie du chameau » en japonais)
- disponible sur Fedora

Python 2.2 (2001)

- Quoi de neuf ?
  - générateur = fonction renvoyant un itérateur

```
#!/usr/bin/python

def suite(x,y):
        print 'entre dans suite'
        while 1:
                yield x
                print 'incremente x'
                x = x + y

s = suite(12,2)
print "ici"
print s.next()
print s.next()
print s.next()
print s.next()
```

```
ici
entre dans suite
12
incremente x
14
incremente x
16
incremente x
18
```

Python 2.2 (2001)

```
#!/usr/bin/python

def fact():
        n = 1
        f = 1
        while 1:
                f *= n
                yield f
                n += 1

for i in fact():
        print i
         if i>1000000: break
```

```
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
```

Python 3 vs Python 2.x

- des incompatibilités
  - les fonctions renvoient des itérateurs plutôt que des listes
  - changement de comportement dans les conditions
  - changement de codage interne (unicode)
  - changement dans la sémantique de la division /
  - changement dans la syntaxe

- comment migrer ?
  - porter le script sous python 2.6
  - exécuter le script avec l'option -3 et corriger les warnings
  - exécuter le traducteur 2to3, exécuter le résultat sous python 3, et règler les problèmes restants

Ruby en 2011

- version stable 1.9.2 depuis août 2010

Conclusion : quel langage utiliser en 2011 ?

Conclusion : quel langage utiliser en 2011 ?

La programmation, c'est un peu comme la musique :

Conclusion : quel langage utiliser en 2011 ?

La programmation, c'est un peu comme la musique :

laissons à l'artiste le choix de l'instrument !