M A S T E R 1

Semestre 2

INTRODUCTION AU LANGAGE PERL

UNIVERSITÉ DE LA SORBONNE NOUVELLE

INALCO

Site du cours : http://icampus.univ-paris3.fr/course/view.php?id=661

Plan du cours

Introduction

Cours L6T55 : Programmation pour le TAL

> Transparents pour le cours > perl-now-and-then > ouvrir index.html avec firefox

Objectifs de cette initiation à pearl : (lire un fichier avec Perl)

- savoir écrire un programme de concordance

- savoir écrire un programme de segmentation d'un fichier en mots

- dans les deux cas, il s'agit de lire un fichier pour :

- extraire des contextes autour d'un pôle choisi

- construire le dictionnaire de tous les mots contenus dans le fichier

Découvrir Perl

Version

→ évolue très vite depuis 2007 mais 5.8.9 suffit (= version stable).

Sur windows à la fac : 5.10.1

Si on travaille :

- sur **ubuntu** : il y a une version installée par défaut. On peut s'en contenter quand on débute. Il ne faut jamais la désinstaller mais on peut en installer d'autres.
- sur **windows** : il faut installer la toute dernière version de perl sur Active State (activestate.com) : elle est fournie avec un outil bien fait : le gestionnaire d'outil pour perl (permet de gérer toutes les bibliothèques disponibles dans l'environnement de perl et cela à travers une interface graphique).

→ Lire le document «Installation de module perl CPAN» pour les bibliothèques.

Pour connaître la version de perl : dans le terminal : **perl -v**

Lancer un script Perl

En général les scripts perl ont pour extension .pl

Si on a plusieurs versions de perl d'installées sur la machine, c'est la plus récente qui va être lancée automatiquement. Pour lancer un script : utiliser la syntaxe générale pour lancer une commande unix :

perl nom_du_programme_à_exécuter + [argument_du_script1, argument_du_script1...]

(perl script.pl fichier.txt)

Les arguments du script ne sont pas obligatoires, c'est une précision.

Mais il y a d'autres façons ...

Les bases à connaître pour programmer en perl

- Notion de **script** (perl en unilingue : non vu en cours cf slides).

- Perl et les **interfaces graphiques**? (Le Trameur / mkAlign).

- Les **scalaires** (commence toujours par $).

- Les **tableaux** / listes (noté @).

- Les tables de **hachage** ou tableaux associatifs (noté %).

- **Structure de contrôle** et **expression conditionnelle** (if, while...).

- Entrée / sortie : lire / écrire dans un **fichier.**

- **Parcours** de fichier.

- L'utilisation des **expressions régulières** en perl (spécificité de perl dans le passé).

→ Écrire un code le plus lisible possible avec le moins de raccourci possible.

Caractéristiques

- À l'origine, langage de **glue.**

- Intermédiaire entre le shell, awk, sed d'un côté et C de l'autre.

- Ajouts de langages fonctionnels comme **Lisp**.

- Maintenant, langage **dynamique généraliste**.

- Extrêmement **portable**.

- Très **stable**.

Slogan

There is more than one way to do it (TIMTOWTDI).

Les trois vertus du programmeur

Paresse, impatience et orgueil.

Points forts de perl

- **Multi-paradigmes** : impératif, fonctionnel, orienté objet.

- Types de base : **scalaire** (chaîne, nombre), **tableau**, **hash**.

- Gestion de mémoire par **ramasse miettes**.

- **Expressions régulières**.

- **OO** : multi-héritage, surcharge d'opérateur, fermetures.

- **Unicode**.

Points faibles

- **Faiblement typé** : en perl on n'a pas besoin de prédéfinir le type de variable qu'on va utiliser (pas besoin de dire «j'attend un type chaîne de caractère»).

- **Trop permissif** : ne dit pas «vous êtes en train d'utiliser une chaine de caractère alors qu'on attend un type numérique.

- **Paradigme objet** très (trop) **simple.**

- Ramasse-miettes par comptage de références.

- **Syntaxe** devenue **lourde** pour certaines opérations.

Premier script

#!/usr/bin/perl;

#! : shebang : en tête du fichier en anglais : chaîne de deux caractères qui renseignent l'interpréteur que cette ligne là indique la version (ici de perl) que l'on va utiliser.

→ Bien commenter ses scripts

#je suis un commentaire qui ne dit pas grand chose...;

print «Hello, word\n»;

!!!!!!!!!!!!!!!!!!!!!!!!! Toutes les commandes perl se terminent par un «;» !!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Commande **print** : affiche quelque_part quelque_chose si on ne précise pas où, ça sous-entend que ça l'affiche à l'écran.

\n : retour à la ligne (obligatoire sinon pas de retour à la ligne entre le résultat et la prochaine ligne)

Pour l'exécuter dans le terminal : perl script.pl

Donne : Hello, world

Sous ubuntu on peut lancer le script d'une autre façon : rendre le script exécutable avec **chmod +x script.pl puis ./script.pl**

Trois structures de données en perl

Variable scalaire

Les structure de données de base en perl permettent de contenir soit des variables numériques, soit des variables de type chaîne de caractère. Leur nom commence obligatoirement par $: $var.

Script2 :

$var = «du texte au hasard»;

print «var=$var\n»; # «var=du texte au hasard»

$var = 42;

print «var=$var\n»; # «var=42»

→ Cela ne pose pas de problème d'assigner une valeur chaîne de caractère puis numérique à une variable en perl : dans l'éditeur de texte .pl, perl reconnaît la différence et les affiche dans des couleurs différentes.

Liste[1] / Array[1] / Tableau [1]

@ définit les listes

→ quand on définit une liste, les éléments de la liste sont des scalaires

Script 3

@array = (1, 2, 3, 4, 5);

@array = («abc», «def», «ghi»);

→ Si on veut que le 3e élément de la liste : (l'indexation des éléments de la liste commence à 0).

$array[2] : $ nom_de_la_liste[position_de_l'élément_dans_la_liste]

script3_timtowtdi

@array = (1 .. 5);

@array («a» .. «z»);

@array qw(abc def ghi);

[1] de scalaires

Valeurs et tranches

$var = $array[2]; #3

@values = @array[1, 3]; #(2, 4)

Fonctions

Push

Elle permet d'ajouter des éléments à la fin d'une liste. On l'écrira avec des parenthèses pour plus de clarté :

push(nom_liste, éléments_à_ajouter)

push (@array, 6, 7)

Pop

Enlève le dernier élément de la liste

Shift

Supprime l'élément en première position dans la liste

Push

Ajoute un élément au début de la liste

Delete

Supprime un élément de la liste

→ On utilisera le plus souvent **shift** et **push** : empiler et dépiler par la gauche.

Tableau associatif [2] / hashage [2]

Un tableau associatif contient des clés. Chaque clé contient une valeur. C'est l'utilisateur qui crée l'association clé-valeur.

[2] de scalaires

CLÉ	VALEUR
«be»	2
«SF»	«prof»
10	«dix»

%hash = (

laz ⟹ «Linux Azur»,

jm2l ⟹ «Journée Méditerranéenne des Logiciels Libres»,

42 ⟹ «la réponse»,

) ;

laz est une clé, sa valeur est «Linux Azur»

%tableau associatif

Keys : on prend toutes les clés, on les met dans une liste.

Values : on cherche toutes les valeurs et on les met dans une liste.

Exists : va chercher à savoir si la clé / la valeur existe.

Création d'une table de hashage

Liste implicite de toutes les clés - valeur.

Bonne structure pour mémoriser pour chaque mot sa fréquence.

Parcours de liste / de hash

Foreach

Parcours une liste ou une table de hashage.

Lire une liste

```
foreach my $element (@array) {     #Pour chacun des éléments de la liste...

#on fait quelque chose avec l'élément de la liste

print «$element est l'élément n° ??? de la liste ! \n»;                #Comment modifier ce code pour récupérer le n°?

}
```

Lire un tableau associatif

→ On parcours la table de hashage par ses clés (keys).

```
foreach my $element (keys %thearray) {  # Pour chacun des éléments de la liste des clés...

        $i++;              #on incrémente le n° de l'élément

        print $hash {$element}, « est l'élément n°$i de la liste ! \n»;

foreach my $a (keys %array) {       # Permet de récupérer les clés du tableau

print «la clé $a a pour valeur ($array\$a)»;
```

While

Expression conditionnelle.

Idée : considérer la table de hashage comme une liste de couples clé-valeur.

each : dans la boucle while permet d'appréhender cette table de hashage comme une liste de couples clé-valeur. Tant que le programme peut créer un couple, il continue.

```
While (($key, $val) = each %thearray) {    #la fonction each permet de récupérer les couples clé-valeur

#on fait quelque chose avec la clé et/ou la valeur de la clé.
```

Lire / Ecrire

Pour lire/écrire un fichier en perl il faut l'ouvrir > script7.pl

Open/Close

Open

FILE : descripteur de fichier / pointeur de fichier (il peut être un scalaire).

Ouverture d'un fichier en lecture

open(FILE1,toto.txt);

open my $fh1, «<« , «toto.txt» ; #notation perl moderne : open a 3 arguments

Ouverture d'un fichier en écriture

open(FILE2, «>titi.txt»);

open my $fh2, «>« , »titi.txt» ;

Close

close(FILE);

Lecture

→ Par défaut, la lecture d'un fichier en perl se fait ligne par ligne (par retour à la ligne : \n). On peut changer en lecture par paragraphe par ex. : \p mais pas dans ce cours.

En perl il y a une variable scalaire prédéfinie $\

$@=<IN>; #contient la première ligne du fichier

$@=<IN>; #contient la deuxième ligne

...

@ARGV : contient la liste des arguments passés à un programme (> script8)

Expressions régulières

Exercice : ouvrir un fichier et écrire les lignes qui contiennent un certain mot dans un autre fichier. Ici, c'est la chaîne de caractères «chat».

→ On interroge chacune des lignes : «Est-ce que *$ligne* contient la chaîne de caractères «chat»? Si oui, on écrit la ligne dans un autre fichier. Sinon, rien.» On utilise le programme idiot de tout à l'heure (script 8) → script 10.pl

$` → contexte gauche

$& → mot reconnu dans la chaîne de recherche

$' → contexte droit

Exercice : créer un dictionnaire qui contient des mots >script11.pl

Rechercher/Remplacer

Supprimer = rechercher et remplacer par rien.

Slide 29 ligne 6 on recherche un motif (ici une balise) et on la remplace par rien, si on met pas g ... (bref il faut le mettre).

Le «my» est un opérateur qui permet de limiter la portée d'existence d'une variable → la variable $mot n'existe que dans certaines zones du programme.

Bibliographie

- http://www.maddingue.free.fr/conference/jm2l-2009/perl-now-and-then/