Administration Réseaux et Systèmes

II. Expression Relationnelle et Programmation Shell

Sommaire

Expression Relationnelle Présentation

- Les expressions rationnelles sont des motifs (pattern en anglais) permettant de décrire des chaîne de caractères.
- Pour faire quoi ?
  - Recherches (**grep**(1)) ;
  - Substitutions (**sed**(1)).
- Plus évolués que les jokers (pattern matching) car elles gèrent les répétitions, les captures et les références arrières.
- Utilisable avec de nombreuses commandes dans le shell et avec la plupart des langages de programmation (exemple : analyse syntaxique).

Expression Relationnelle Présentation

- Il existe plusieurs variantes d'expressions rationnelles :
  - Expression relationnelles standards ;
  - Expression relationnelles étendues ;
  - Expression relationnelles compatibles perl(1) (PCRE).
- Dans ce cours, nous nous intéresserons aux *expressions rationnelles étendues*.

Expression Relationnelle
Notion d’atomes

Décrire un caractère :

a	Le caractère «a»
.	N’importe quel caractère (joker);
[aeiouy]	Un caractère dans la liste : une voyelle minuscule;
[ˆaeiouy]	Un caractère hors de la liste : une pas-voyelle-minus- cule
[a-y]	Un caractère dans l’intervalle : une minuscule qui n’est pas « z »

Expression Relationnelle
Notion d’atomes

Possibilité d’utiliser des ensembles prédéfinis

[:alnum:]	Un caractère alphanumériques
[:digit:]	Un chiffre
[:xdigit:]	Un chiffre hexadécimal
[:lower:]	Une minuscule
[:upper:]	Une majuscule
[:alpha:]	Une lettre
[:space:]	Un caractère d’espacement (espaces, tabulation, etc)

Les ensembles devant être spécifiés entre crochets, il y a bien deux crochets.

Exercice : quelle expression rationnelle correspondra aux différentes tailles standard de papier, de « Format A0 » à « Format A9 » ?

Expression Relationnelle
Répétition d'atomes

Les répétitions suivent les atomes pour indiquer combien de fois l'atome se répète dans le motif :

?	L'atome est présent 0 ou 1 fois
*****	L'atome est présent 0 ou n fois
+	L'atome est présent 1 fois ou plus
{n}	L'atome est présent n fois
{n,}	L'atome est présent au moins n fois
{,m}	L'atome est présent au plus m fois
{n,m}	L'atome est présent entre n et m fois

Exercice : quelle expression rationnelle correspondra à un mot (une suite d'au moins un caractère alphabétique situé entre deux espaces) ?

Expression Relationnelle
Quelques caractère spéciaux

^	Début d'une ligne
$	Fin d'une ligne
\<	Début d'un mot
\>	Fin d'un mot
\b	Début ou fin d'un mot

Pour rendre un sens normal à un caractère spécial (comme « . », « ˆ » ou « $ »), on l'échappe avec un « \ ».

Exercice : quelle expression rationnelle correspondra à une suite de 0 ou plusieurs « * » en début de ligne et au moins un « + » en fin de ligne ?

Expression Relationnelle TDs

Exercice 01 :

Décrivez une expression rationnelle qui corresponde à une ancienne plaque d'immatriculation française de métropole (du type « 1788 VD 63 ») :

Un nombre compris entre 1 et 9999 (un chiffre entre 1 et 9 suivi d'au plus 3 chiffres entre 0 et 9) ;

1 à 3 lettres en majuscule ;

2 chiffres pour le département (le département « 00 » n'est pas valide, pour l'instant nous l'accepterons).

Exercice 02

Décrivez une expression rationnelle qui corresponde à un nombre relatif (« 1.5 », « +37.5 », « -0.7 », etc) :

Un signe facultatif ;
Au moins un chiffre (partie entière) ; Un point facultatif ;
Au moins un chiffre (partie décimale).

Cette expression correspond à des entrées qui ne sont pas des nombres relatifs. Donnez des exemples.

Expression Relationnelle Alternatives

Décrire plusieurs possibilités :

(foo | bar |….)

Un « | » indique un «ou».

On peut améliorer l'expression qui correspond aux départements des plaques d'immatriculations de manière à rechercher :

- Un « 0 » suivi d'un chi re différent de « 0 »;
- Un chiffre différent de zéro suivi d'un chiffre.

De même, on peut améliorer l'expression qui correspond aux nombres relatifs pour rechercher :

- Un nombre entier ;
  Un nombre avec une partie fractionnaire.

Expression Relationnelle Répétitions de Blocs

Un bloc entre parenthèse peut être répété de manière similaire aux atomes.

(bloc) ?

Un bloc peut être présent zéro ou une fois.

(bloc) *

Un bloc peut être présent zéro fois ou plus.

(bloc) +

Un bloc peut être présent une fois ou plus.

(bloc) {3}

Un bloc peut être présent trois fois.

Expression Relationnelle
Références arrières

Les blocs entres parenthèse font l'objet de captures.
Il est possible de faire référence à un texte capturé pour indiquer des répétitions.

([[:digit:]]{4}).*\1

Correspond aux lignes qui contiennent 2 fois la même suite de 4 chiffres.

(.)(.)(.).*\3\2\1

Correspond à trois caractères qui se suivent dans un sens puis dans l'autre sur la même ligne (les...sel).

Expression Relationnelle
Outils en lignes de commande

grep(1) Affiche les lignes corsesondant à un motif donné

Nous utiliserons avec son option -E

Sed(1) Éditeur de flux permettant le filtrage et la transformation de texte. Nous utiliserons avec son option -r et essentiellement dans sa syntaxe pour faire des substitutions de texte

Ex: % sed -re ’s/([0-9]+)/[\1]/g'

Awk Filtre programmable

Les quotes
Présentation

Lorsque l'on exécute une commande, par exemple :

% echo a b c

le shell l'analyse pour extraire le programme à lancer (ici echo(1)) et les arguments à lui passer (ici, « a », « b » et « c »), le séparateur étant une suite de caractères « blancs ».
Dans notre cas, **echo**(1) est exécuté avec 3 arguments qu'il affiche en les séparant avec un espace :

a b c

Les quotes vont notamment permettre de :

- Passer des arguments qui contiennent des blancs ;
- Passer des chaînes vides comme arguments.

Les quotes

Guillemets droits simples (simple quotes)

Le texte entre simple quotes ('...') est traité littéralement, sans aucune interprétation.

```
% echo   ‘Hello'                  # 1 seul argument passé à echo (1)
Hello

% echo a b     c                  # 3 arguments passés à echo(1)
a b c

% echo 'a   b    c'                # 1 seul argument passé à echo(1)
 a   b    c

% echo '$$ `a`'                    # 1 seul argument passé à echo(1)
$$`a`
```

Les quotes

Double Guillemets simples (double quote)

Le texte entre double quote ("...") est traité littéralement, sauf les caractères« " »,« $ »et« ` »:

```
% echo "Hello"
Hello
% echo "Hello \"World\""
Hello "World"
% a=toto
% echo "a=$a"
a=toto
% echo "Vous utilisez `uname`"
Vous utilisez FreeBSD
```

Les quotes

Guillemets droits inversés (backquote)

Si une commande contient des expressions entre backquote (`...`), le shell les évalue et les substitue par leurs résultats avant de l'exécuter :

% echo `uname -s` `uname -r` #=> echo FreeBSD 10.2-STABLE

FreeBSD 10.2-STABLE

La syntaxe $(...) est équivalente. Un peu plus lisible, elle permet les imbrications :

% echo $(uname -s) $(uname -r) #=> echo FreeBSD 10.2-STABLE

FreeBSD 10.2-STABLE

Les quotes
Exemples

Que font les commandes suivantes :

```
a=$(date)
b=$(ls ~ | grep '\.[ch]$' | wc -l)
c='$PATH est la liste des répertoires des exécutables'
d="nous sommes le $a"
e="Nous sommes le $(date)"
f=$($d)
g='1234 5'
echo $g
echo "$g"
cc=`which clang`
cc=$cc' -Wall -g -O0'
if $($cc "mon-prog.c" -o mon-prog); then echo OK; fi
```

Script Shell
Présentation

Un script est un fichier texte contentant une **succession de commandes shell**.

Le fichier doit être rendu **exécutable** (chmod(1)) avant de pouvoir être exécuté comme n'importe quelle commande.

La première ligne contient le *shebang* qui indique l'interpréteur, par exemple« #!/bin/sh »,«#!/usr/bin/env ruby».

Le système reconnaît les deux premiers octets « », il exécute alors la commande qui suit et lui passe en paramètre le chemin du script.

Script Shell

Arguments des scripts

Le shell met quelques variables à disposition pour gérer les arguments passés à un script :

$1	Premier argument
$2	Second argument
.	
$9	Neuvième argument
$0	Commande exécutée
$#	Nombre d'arguments
$* / $@	Tous les arguments

La commande interne **shift**(1) supprime le premier argument et décale les autres ($2 devient $1, $3 devient $2, etc).

Script Shell

Arguments des scripts

```
% nano script.sh
# !/bin/sh
echo "Je suis $0, \$2=$2 \$1=$1"

% chmod +x script.sh
% ./script.sh foo bar
Je suis ./script.sh $2=bar $1=foo
```

Script Shell

Évaluation d'expressions avec [/ test

Lors de l'écriture d'un script, il est souvent nécessaire de tester si des conditions sont remplies. Pour cela, on utilise la commande [(1) ou test(1).

% [expression]

% test expression

Où expression vérifie des assertions sur des fichiers ou effectue des comparaisons

Si expression est vérifiée, la commande retourne 0, sinon elle retourne 1.

Script Shell

Évaluation d’expressions avec [/ test

Expressions de comparaisons :

-n STR	La longueur de STR est non nulle
STR	Équivalent à -n STR
-z STR	La longueur de la STR est nulle
STR1 = STR2	Les deux chaines sont égales
STR1 < STR2	STR1 vient avant STR2 par ordre alphabétique;
STR1 > STR2	STR1 vient après STR2 par ordre alphabétique;
STR1 != STR2	Les deux chaines sont différéntes
INT1 -eq INT2	INT1 et INT2 sont égaux
INT1 -ge INT2	INT1 est supérieur ou égal à INT2
INT1 -gt INT2	INT1 est strictement supérieur à INT2
INT1 -le INT2	INT1 est inférieur ou égal à INT2
INT1 -lt INT2	INT1 est strictement inférieur à INT2
INT1 -ne INT2	INT1 et INT2 sont différénts

Script Shell

Évaluation d'expressions avec [/ test

Expressions d'assertions sur les fichiers:

FILE1 -ef FILE2	FILE1 et FILE2 sont le même fichier
FILE1 -nt FILE2	FILE1 est plus récent que FILE2
FILE1 -ot FILE2	FILE1 est plus vieux que FILE2
-e FILE	FILE existe
-f FILE	FILE existe, c'est un fichier ordinaire
-d FILE	FILE existe, c'est un répertoire
-r FILE	FILE existe, et ses permissions permettent de le lire
-w FILE	FILE existe, et ses permissions permettent de l'écrire
-x FILE	FILE existe, et ses permissions permettent de l'exécuter
-s FILE	FILE existe, et il a une taille non nulle

Script Shell

Évaluation d'expressions avec [/ test

Expressions sur les expressions:

(EXP)	EXP est vérifiée
! EXP	EXP n'est pas vérifiée
EXP1 -a EXP2	EXP1 et EXP2 sont vérifiées
EXP1 -o EXP2	EXP1 ou EXP2 est vérifiée

Par souci de lisibilité, certaines personnes préfèrent combiner plusieurs appels à [(1) plutôt que de combiner les expressions. Les lignes suivantes sont équivalentes :

```
[ 0 -le $a -a $a -le 10 -o $a -eq 12 ]
```

```
[ 0 -le $a ] && [ $a -le 10 ] || [ $a -eq 12 ]
```

Script Shell
Évaluations arithmétiques

La commande expr(1) permet d'évaluer des expressions arithmétiques (attention aux espaces indispensables pour séparer opérateurs et opérandes) :

% i=`expr $i + 1`

La plupart des shell proposent une syntaxe plus pratique (pas sensible aux espaces) :

% i=$(($i + 1))

Note: expr(1) étant un programme externe, il est beaucoup plus lent que la syntaxe $((...)) .

Pour les calculs en flottants, on utilisera bc(1) :

% r=`echo « scale=8; sqrt(2)" | bc`

Script Shell

Lecture sur entrée standard

read(1) est une commande interne permettant de lire des chaînes sur l'entrée standard : read [nom ...] .

Exemple :

printf "Entrez votre nom : "

read NAME JUNK

echo "Bonjour, $NAME"

Si des noms de variables sont précisés, le premier mot est stocké dans la première variable, le second dans la deuxième, et ainsi de suite. Les mots excédentaires sont enregistrés dans la dernière variable.

Script Shell

Structure de contrôle

if ... then ... fi

if COMMANDES; **then** COMMANDES;
[**elif** COMMANDES; **then** COMMANDES;]...
[**else** COMMANDES;]
fi

La liste ***if COMMANDES*** est exécutée. Si elle retourne 0, la liste ***then COMMANDES*** est exécutée et la commande if se termine. Dans le cas contraire, chaque liste ***elif COMMANDES*** est exécutée à son tour et si son code de retour est 0, la liste ***then COMMANDES*** correspondante est exécutée et la commande se termine. Sinon, la liste ***else COMMANDES*** est exécutée si elle existe.

Script Shell

Structure de contrôle

if ... then ... fi : Exemples

```
if  cp  foo  /tmp; then echo "Fichier copié avec succès"; fi
```

```
if grep -q 's1' file1 || grep -q 's1' file2; then echo "Trouvé dans file1 ou file2"
elif grep -q 's2' file3; then echo "Trouvé dans file3"
fi

if [ ! -e file ]; then echo "N'existe pas"; fi
```

Script Shell

Structure de contrôle

if ... then ... fi : Exemple de script

```
#!/bin/sh
if [ "$#" -ne 1 ]; then
    echo "usage:  $0  fichier" >&2
    exit 1
fi
if [ -f "$1" ]; then
    echo "C'est un fichier"
    ls -l "$1"
else
    echo "Mais qu’est-ce ?"
fi
```

Script Shell

Structure de contrôle

if ... then ... fi : TD

1. Écrire un script qui reçoit en paramètre l'heure (3 arguments : heures, minutes, secondes) et qui retourne :

   ➡ 0 si l'heure est au bon format (0 ≤ heures ≤ 23, 0 ≤ minutes ≤ 59 et 0 ≤ secondes ≤ 59) ;

   ➡ 1 sinon et affiche sur la sortie d'erreur "Format incorrect".

2. Écrire un script qui retourne 0 si le nombre passé en paramètre est pair, 1 sinon.

Script Shell

Structure de contrôle

Les Boucles :
Exécuter (Répéter) des commandes aussi longtemps qu'un test réussit :

% while COMMANDES; do COMMANDES; done

Exécuter des commandes aussi longtemps qu'un test échoue :

% until COMMANDES; do COMMANDES; done

Exécuter des commandes pour chaque membre d'une liste :

% for NOM [in MOTS ...]; do COMMANDES; done

Pour chaque mot, ***COMMANDES*** sont exécutées, le mot actuel étant accessible dans la variable **$NOM.**

Si aucune liste n'est précisée, for itère sur les paramètres du script (« $@")

Script Shell

Structure de contrôle

Les Boucles :Exemples

```
i=0;
until [ $i -eq 10 ]; do i=$(($i+1))
done

for fruit in pomme poire banane; do echo "J'aime␣les␣${fruit}s" ;
done

for i in `seq 1 10`; do echo $i
done
```

Script Shell

Structure de contrôle

Les boucles: TD

1. Écrire un script effectuant la copie de tous les fichiers d'un répertoire passé en paramètre dans le répertoire courant ;

2. Écrire un script de chronométrage : il « compte » les secondes puis les minutes puis les heures dans 3 boucles imbriquées et affiche chaque seconde le temps écoulé depuis son lancement. Utilisez 3 types de boucles.

Script Shell

Structure de contrôle

Les boucles: TDs suite

1. Écrivez un script acceptant en argument une liste d'entiers et qui n'affiche que ceux qui sont positifs ou nuls ;

2. Écrivez un script lisant son entrée standard et n'affichant que les lignes paires ;

- qui en plus contiennent un nombre d'au moins deux chiffres ;
- en les passant en majuscules (via la commande tr a-z A-Z) ;
- en les faisant précéder de leur numéro (1 pour la première ligne).

3. Écrivez un script qui lit des notes sur l'entrée standard, et affiche la moyenne sur la sortie standard.

FIN COURS 3