Chapitre 5

Outils de traitement de données

1. Introduction
2. Structure des Expressions Régulières
3. Awk

1. Introduction

- Expression régulière :
  - Suite de (méta-)caractères pour désigner une "famille" de chaînes de caractères
  - Permet de reconnaître des "formes" dans un texte, afin
    - de les repérer, compter, …
    - de les extraire, recopier, sauver ailleurs, …
    - de les détruire, corriger, transformer, …

Mode d'Utilisation

- Les utilitaires qui utilisent les expressions régulières travaillent en "mode ligne"
- La ligne dans un fichier UNIX (suite d'octets)
  - Est délimitée par le caractère "nouvelle ligne"
    - alias "retour chariot",
    - alias "saut de ligne" ...
  - Le caractère "nouvelle ligne" ne fait pas partie de la ligne à traiter

Expressions Basiques et Etendues

- Basic Regular Expressions (BRE)
  - Historiquement les premières
  - Protection systématique des méta-caractères avec \
    - Source d'erreur (on en oublie toujours)
    - Difficile à lire : que fait \([ab]\?\\\{2\}\)\1 ?
  - Offre une bizarrerie : la mémoire de rappel
  - Outils : ed, grep, sed
- Extended Regular Expression (ERE)
  - Nouveau standard POSIX
  - Pas besoin de protéger les méta-caractères
  - Plus fidèles au concept théorique des regexp
    - Disparition de la mémoire de rappel
  - Outils : egrep (grep –E), (g)awk

2. Structure des Expressions Régulières

- Structure

ERE = branch|branch|…

branch = piece piece…

piece = atom

= atom*

= atom+

= atom?

= atom bound

bound = {n}

= {n,}

= {n,m}

BRE =pieceb pieceb …

pieceb = atomb

= atomb*

= atomb\+

= atomb\?

= atomb boundb

boundb = \{n\}

= \{n,\}

= \{n,m\}

Structure des Expressions Régulières

- Dans les définitions qui suivent,
  - le caractère '**c**' exclut le caractère "nouvelle ligne" (noté '**nl**')
  - le caractère '**d**' désigne un digit

- Structure (suite...)

  atom = **(**ERE**)**
  = **()**
  = bracket
  = item

  atomb = **\(**BRE**\)**
  = **\(\)**
  = bracket
  = item
  = **\d**
  = **\<**
  = **\>**

Structure des Expressions Régulières

Les 'item' (communs aux ERE et BRE) :

- **\c** un '\' suivi du caractère 'c' désigne ce caractère
- **^** symbolise le début de ligne
- **$** symbolise la fin de ligne
- **.** symbolise n'importe quel caractère
- **c** un caractère sans signification particulière se désigne lui-même
- **\w** un caractère **dans** un mot
- **\W** un caractère **en dehors** d'un mot

Structure des Expressions Régulières

- Les 'brackets' (communs aux ERE et BRE)
  - **[c_1c_2…]** Un caractère (un seul) parmi ceux qui sont entre crochets (**c_1c_2…**)
  - **[^c_1c_2…]** Un caractère (un seul) **sauf** ceux qui sont entre crochets
  - **[c_1-c_2] [^c_1-c_2]** La notation **c_1-c_2** désigne l'ensemble des caractères qui sont dans l'intervalle qui va du caractère **c_1** au caractère **c_2**
  - **[[.c_1c_2….]]** La suite consécutive des caractères **c_1c_2…** (utile ?)
  - **[[=c=]]** Un caractère **équivalent** au caractère **c**
    ex: [[=e=]] équivalent à [eéèêë]
  - **[[:<:]]** Symbolise un début de mot (idem **\<** dans BRE)
  - **[[:>:]]** Symbolise une fin de mot (idem **\>** dans BRE)
  - **[[:**class**:]]** Un caractère parmi ceux de la classe class

Structure des Expressions Régulières

- Les classes de caractères ('class')
  - **alnum** : alphabétique ou numérique
  - **alpha** : alphabétique
  - **blank, space** : espace ou tabulation
  - **cntrl** : caractère de contrôle
  - **digit, xdigit** : chiffre (décimal ou héxa)
  - **graph** : caractère graphique
  - **lower, upper** : caractère alpha minuscule ou majuscule
  - **punct** : caractère de ponctuation

Structure des Expressions Régulières

- Mémoire de rappel des BRE
  - Chaque bloc de parenthèses est associé à une mémoire
  - Le texte correspondant (match) à la sous-ER entre parenthèses est sauvé dans la mémoire
  - Les mémoires sont numérotées selon l'ordre d'apparition des parenthèses ouvrantes (9 au maximum
  - Les mémoires peuvent être rappelées à l'aide de l'atome **\d** (cad \1, \2, …, \9)
- Exemple : \([abc]\)\1 désigne aa, bb ou cc
- Attention : ce mécanisme peut ralentir considérablement l'évaluation…

3. Awk

- Concepts généraux
  - AWK est une extension de grep
    - Un langage de programmation "à la C" permet d'exécuter des traitements sur les lignes d'un fichier
    - Plusieurs traitements différents peuvent être définis
    - Le filtrage (notamment par ERE, mais pas seulement) détermine sur quelles lignes d'un fichier appliquer quel(s) traitement(s)
  - AWK est un filtre qui
    - permet le balayage de très gros fichiers
    - opère au moyen de couples sélection/action

Quelques Caractéristiques

- AWK
  - Reconnaît les ERE
  - Distingue les lignes et les champs dans une ligne
  - Offre des possibilités de calcul courant
    - Structures de contrôle à la C : if () {} else {}, boucles for (;;) {}, while () {}, ...
    - Fonctions d'affichage (printf), extraction de chaines, ...
  - Permet de coder des procédures pré et post-opératoires

Utilisation

1. $ awk –f cmdfile fich1 fich2 …
   - cmdfile est fichier contenant le programme du traitement
   - Les noms des fichiers de données à traiter sont donnés à la suite
     - Les contenus des fichiers sont concaténés
   - Le résultat du traitement est placé sur stdout
2. $ awk [-e] 'programme' > fich_out < fich_in
   - En l'absence de nom de fichier de données, AWK traite l'entrée standard

Structure d'un Programme AWK

BEGIN { instructions de début }
sélecteur 1 { actions associées au sélecteur 1 }
sélecteur 2 { actions associées au sélecteur 2 }
...
sélecteur n { actions associées au sélecteur n }
END { instructions de fin }

function fname(a,b,c ...) { corps de la fonction }
function fname2 ...

Règles Concernant les Sélecteurs

- Un sélecteur peut être
  - Une ERE
  - Une expression logique
  - Une combinaison d'ER et d'EL
  - Une expression d'intervalle (délimitée par deux sélecteurs)
- Le sélecteur est optionnel
  - En cas d'absence toutes les lignes sont sélectionnées
    - autrement dit, le traitement est systématiquement appliqué à chacune des lignes de données lues

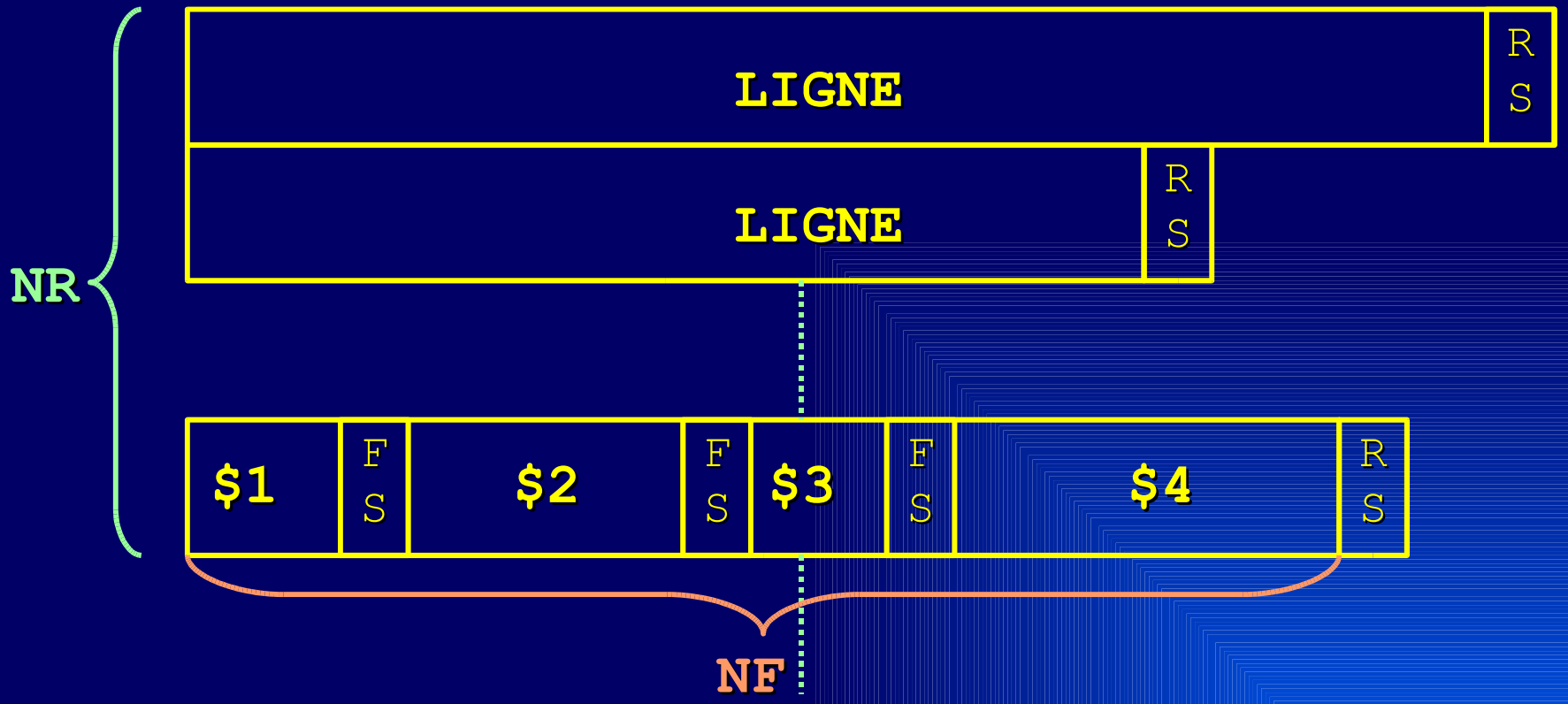
Règles Concernant les Actions

- Une action est une suite d'instructions
- L'action a plusieurs possibilités de sortie
- L'action est optionnelle
  - En cas d'absence, la ligne sélectionnée est affichée sur la sortie standard

Algorithme d'AWK

- Concept
  - La prochaine ligne de données est lue
  - La ligne lue est comparée successivement à chaque sélecteur
  - Si la sélection convient, l'action correspondante est effectuée
- Boucles
  - Première boucle sur chaque ligne
  - Deuxième boucle sur chaque sélection/action
  - Deux exceptions : sélecteurs BEGIN et END
    - BEGIN : l'action est réalisée avant la lecture de la première ligne
    - END : l'action est réalisée après le traitement de la dernière ligne

Analyse des Données : Lignes et Champs



Quelques Variables Utiles

- Vocabulaire
  - Les lettres R, F, S, et O signifient
    - ligne (Record), champ (Field), séparateur (Separator), sortie (Output)
- NR, NF : nombre courant de lignes, de champs
- FS, RS : délimiteur de lignes, de champs
- OFS, ORS : délimiteur de lignes, de champs, mais en sortie
- $0, $1, … $n : contenu de la ligne courante, de champ 1, …, du champ n

Construction des Sélecteurs

- Opérateurs de comparaisons
  - >, <, >=, <=, ==, !=, ~, !~
- Opérateurs logiques
  - || (ou), && (et), ! (non), exp1?exp2:exp3
- Délimiteur d'expression régulière : '**/**'
  - ex:
    - /^abc/ = "qui commence par abc"
    - différent de 'abc' = variable de nom abc !
- Expression d'intervalle : un sélecteur composé de 2 sous-sélecteurs séparés par une vigule :
  - toutes les lignes comprises entre ls deux lignes correspondantes sont sélectionnées
  - ex: /<html>/,/<\/html>/

Exemples de Sélecteurs

- $1 == “toto” && NF == 4

  Le premier champ est exactement égal à la chaîne “toto” et la ligne comporte 4 champs.

- $1 ~ /toto$/ && $3 ~ /[[:digit:]]/

  Le premier champ se termine par le motif “toto” et le 3e champ contient un chiffre

- $1 ~ /^d/ || substr($7,1,2) > 12

  Le premier champ commence par la lettre 'd' ou les 2 premiers caractères du 7e champ forment un nombre > 12

Construction des Actions

- Une action est une suite d'instructions
- Une instruction est soit :
  - une affectation
  - une instruction conditionnelle
  - une boucle
  - une des fonctions internes
  - une des fonctions définies par l'utilisateur

Construction des Actions

- Exemples de constructions
  - **if (**condition**)** instruction **[else** instruction **]**
  - **while (**condition**)** instruction
  - **for (**expr **;** cond **;** expr **)** instruction
  - **for (**variable **in** tableau**)** instruction
  - **break**
  - **continue**
  - **{** instruction **}**
  - variable**=**expression**;**
  - **next** # ligne suivante (ignorer les sélecteurs restants)
  - **exit** # ne pas traiter le reste des données

Quelques Fonctions Internes

- Impression à l'écran
  - **print** [liste_d'expressions] [> expression]
  - **printf** format [**,** liste_d'expressions] [> expression]
- Travail sur les chaînes
  - **length**(source)
  - **index**(source, caractère)
  - **substr**(source, index, longueur)
  - Et bien d'autres : index, split, sub/gsub, tolower, toupper, …
- Calculs arithmétiques
  - sqrt, log, exp, int, rand, sin, …
- Autres :
  - **system** : exécution dune commande externe
  - **getline** [var] [< file] : forcer la lecture de la ligne suivante
  - **systime, strftime**

Exemples

- Nombre de lignes d'un fichier (wc –l)
  $ awk 'END {print NR}' fich
- 10 premières lignes d'un fichier (head –10)
  $ awk '{print} NR == 10 {exit}' fich
- idem grep –n "mot"
  $ awk '/mot/ {print NR, $0}' fich

Exemples (2)

- Substitution de titi à toutes les occurrences de toto

```
$ awk '/toto/          { for (i=1 ; i<= NF ; i++)
                               if ( $i ~ /^toto$/)
                                        $i="titi"
                       }
                       { print }' < fich.in > fich.out
```

- Inversion d'adresse internet

```
$ awk '$1 ~ /([[:digit:]]{1,3}\.){3}[[:digit:]]{1,3}/ {
                       split($1,adr,".")
                       printf "%s.%s.%s.%s\t%s\n",
                               adr[4], adr[3], adr[2], adr[1], $2
                       }' /etc/hosts
```