

Systèmes d'exploitation

Les expressions régulières

Yves STADLER

Codasystem, UPV-M

23 février 2009

1/19

Introduction

Késako ?

- Chaîne de caractères
- Décrit un ensemble de chaînes de caractères

D'où est-ce que cela vient ?

- De la théorie des mathématiques sur les langages
- Ken Thompson les intègre dans qed -> ed -> grep

3/19

Vocabulaire

Expression régulière / Expression rationnelle

Vocabulaire

Parfois motif, ou en anglais pattern

2/19

Introduction

À quoi ça sert ?

- Décrire et classifier les langages
- Editer, contrôler, manipuler du texte

Comment les utiliser

- Lex
- Expr
- awk
- Perl, TCL, Python

4/19

Les bases

- On veut décrire un ensemble de chaînes par un motif
- On va utiliser des alternatives et des quantifieurs
- On rajoutera quelques méta-caractères

5/19

Usage

- La barre verticale `|` est l'opérateur d'alternation (opérateur d'alternative)
- `a|e` décrit les chaînes suivantes : "a" et "e"
- `ab|eb` : "ab" et "eb"
- `Bonjour|Aurevoir` : "Bonjour" et "Aurevoir"

Usage

- Les parenthèses permettent de gérer la priorité et les groupements
- Exemple avec "ex-équo", "ex-equo", "ex-aequo" et "ex-æquo"
- `(ae|e)quo` et `aequo|equo` désignent tout les deux les chaînes "aequo" et "equo"

6/19

Quantifieurs de bases

0, 1 et plus

- Un quantifieur quantifie le groupement le précédent
- `?` : le groupe peut apparaître zéro ou une fois
- `*` : le groupe peut apparaître zéro, une ou plusieurs fois
- `+` : le groupe doit apparaître au moins une fois, ou plus.

Exemple

- `toto?` désigne : "tot" et "toto"
- `toto*` désigne : "tot", "toto", "totoo", "toto...o"
- `toto+` désigne : "toto", "totoo", "toto...o" mais pas "tot"
- `ex-(a?e|æ|é)quo` désigne : "ex-équo", "ex-equo", "ex-aequo" et "ex-æquo"

7/19

Quantifieurs de bases

0, 1 et plus

- `(a|b)*aaa` désigne toute chaîne commençant par une suite (nulle ou non) de caractères "a" ou "b" se terminant par "aaa"
- `(0|1|2|3|4|5|6|7|8|9)+` désigne tous les nombres entiers (même ceux commençant par des 0)

Attention

- Une chaîne vide peut être décrite par un motif
- `a?b?c?d?e?f?g?h?i?j?k?` décrit la chaîne vide, ou une suite de lettres de a à k se suivant dans l'ordre.
- La chaîne vide correspond (match) aussi le motif `(item)*`.
- En revanche, elle ne match pas le motif `item*`, ni `(item)+`.

8/19

Méta-caractères

Classes

- [...] : N'importe quel caractères dans les crochets
- [^...] : N'importe quel caractères qui n'est pas dans les crochets

Intervalle

- [a-z] : toutes les lettres minuscules
- [A-Z] : toutes les lettres majuscules
- [A-Za-z] : toutes les lettres majuscules et miniscules
- [0-9] : tous les chiffres.

Exemples

- h[aeiouy] désigne : "ha", "hi", "ho", "hu", "hy"
- h[^aeiouy] désigne : "hb", "hc", "hd", "hf" ... sauf "ha", "hi", "ho", "hu", "hy"

9/19

Méta-caractères

Début, fin... joker !

- ^ : début de phrase
- \$: fin de phrase
- . : n'importe quel caractère

Exemples

- ^cours.* : chaîne débutant par cours, et suivie d'autres caractères
- ^cours : chaîne commençant par cours
- cours\$: chaînes se terminant par cours

10/19

POSIX ou PCRE

Vocabulaire

- Beaucoup de syntaxe se sont développées pour les expressions régulières
- Les deux principales sont celles préconisée par POSIX (Portable Open Software Interface X) et les PCRE (Perl Compatible Regular Expression)

Notation

- On commence et termine les expressions PCRE par #

11/19

POSIX

Outils

- ed
- grep
- sed
- vi

Quantifieurs supplémentaires

- {m,n} : m = minimum, n = maximum. (Si pas de premier argument : 0, si le second est omis : infini)
- Parenthèses : ()
- {0,1} ou {,1} : ?
- {0,} : *
- {1,} : +

12/19

POSIX

Attention

- Dans vim ou emacs, les regexp étendue ne sont pas POSIX, il faut échapper les () et { }

Échappement

- (,), [,], ., *, ?, +, ^, |, \$ et \ doivent être échappés pour avoir leur signification textuelle.
- On utilise le backslash.

13/19

Introduction

Vocabulaire

- [:alnum:] Caractère alphanumérique [0-9a-zA-Z]
- [:digit:] Chiffre décimal [0-9]
- [:xdigit:] Chiffre hexadécimal [0-9a-fA-F]
- [:alpha:] Caractère alphabétique [a-zA-Z]
- [:lower:] Lettre minuscule [a-z]
- [:upper:] Lettre capitale [A-Z]

15/19

Classes supplémentaires

Classes POSIX

- [:cntrl:] Caractère de contrôle [\x00-\x1F\x7F]
- [:space:] Espace blanc ou séparateur de ligne ou de paragraphe [\t\r\n\v\f]
- [:blank:] Espace blanc ou tabulation non séparateur de ligne ou de paragraphe [\t]
- [:print:] Espace simple ou caractère graphique visible [\x20-\x7E]
- [:graph:] Caractère graphique visible [\x21-\x7E]
- [:punct:] Caractère de ponctuation [!\"#\$%&'()*+,-./:;?@[_`{|}~]

14/19

PCRE

Quantificateurs

- Comme pour POSIX étendu

Classes

- Dans une classes les caractères ont leur signification littérale.
- Sauf #] -

16/19

Classes abrégées

- `\d` Chiffre : `[0-9]`
- `\D` Pas un chiffre : `[^0-9]`
- `\w` Mot : `[a-zA-Z0-9_]`
- `\W` Pas un mot : `[^a-zA-Z0-9_]`
- `\t` Tabulation
- `\n` Nouvelle ligne
- `\r` Retour chariot
- `\s` Espace blanc (correspond à `\t \n \r`)
- `\S` Pas un espace blanc (`\t \n \r`)

17/19

Comparé à POSIX, PCRE permet

- Plus rapide
- Utilisation des références arrières
- Capture de toutes les occurrences
- Rendre un quantificateur non gourmand
- Utilisation des parenthèses non capturantes
- Utilisation d'une foule d'options
- Utilisation d'assertions, masques conditionnels
- Fonctions de callback

Mais

Nous verrons cela la semaine prochaine ! (Ouf !)

18/19

That's all folks !

Le mot de la fin

MOT