Systèmes d'exploitation

RegExp: Higher and Deeper

Yves Stadler

Codasystem, UPV-M

13 septembre 2011

Objectifs du chapitre

Objectifs

- Savoir quelles commandes peuvent-être utilisées avec des regexp.
- Savoir utiliser la capture et la *backreference*
- Contrôler sa gourmandise
- Rechercher toutes les occurrences

Commandes

POSIX

- ereg (macthing)
- eregi (matching / case insensitive)
- ereg_replace (replacement)
- eregi_replace (case insensitive replacement)
- split (découpe)
- spliti (découpe insensible a la casse)

Commandes

PCRE

- preg_grep — Retourne un tableau avec les résultats de la recherche
- preg_match — Une seule occurence
- preg_match_all — Toutes occurences
- preg_replace — rechercher et remplacer
- preg_replace_callback — Recherche, remplace en utilisant une fonction
- preg_split — Eclatement de chaine

Capture

La parenthèse

- Utiliser une parenthèse créér une pseudo-variable `\i` qui stocke la valeur du sous-motif.

0

Exemple

- `[0-9]{2}-[0-9]{2}-[0-9]{4}` désigne une chaine de type date JJ-MM-AAAA
- `([0-9]{2})-([0-9]{2})-([0-9]{4})`
- `\1` prend la valeur de JJ
- `\2` prend la valeur de MM
- `\3` prend la valeur de AAAA

Capture

Exemple

- `(.*)\1` représente les phrases en deux parties identiques ("pouetpouet", "haha", "toto")
- `t([aeiou])t\1` désigne toutes les chaînes tXtY ou X=Y= une voyelle
- `0[1-6]([0-9]{2}){2}\1\2` désigne les numéros de téléphones qui se terminent par deux groupes de 4 chiffres identiques (06 08 12 08 12, 03 87 70 87 70, ...)

Non capturante, groupements nommés, backreference

PCRE only !

- On peut annuler la capture avec les PCRE (et sans doute aussi dans POSIX étendu)
- `(?:...)`
- Pour utiliser une backreference, il faut au moins le nombre de parenthèses requis.

La gourmandise

Analysons ceci :

- La RE : `.*A.*`
- Et la phrase : "zAzAzAz"
- Comment va réagir la correspondance ?

Possibilités

- .* = z | A | .* = zAzAz
- .* = zAz | A | .* = zAz
- .* = zAzAz | A | .* = zAz

La gourmandise

Est-un vilain défaut

- Dans les RE POSIX, les quantifieurs sont toujours gourmands (Plus c'est long... plus c'est bon)
- Avec les PCRE, les quantifieurs ne sont gourmands que par défaut.

Cure de désintox

- Pour supprimer la gourmandise d'un opérateur on utilise ?
- `.*?A.*`
- Capture : .* = z | A | .* = zAzAz

La gourmandise

Avec PHP

- Fonction ereg()
- `ereg('<a href=(.*)>', $text, $out);`
- `$text="Une phrase avec <a href="http://www.codasystem .com/mapage.php"> que je veux pouvoir matcher</a> mais pas le reste"!`
- out contiendra le résultat de la capture
- POSIX : 'http ://www.codasystem.com/mapage.php"> que je veux pouvoir matcher</a'

PCRE

- Fonction preg_match()
- `preg_match('<a href=(.*?)>', $text, $out);`
- out contiendra le résultat de la capture
- PCRE : "http ://www.codasystem.com/mapage.php"

Assertions

Assertions simples

- \b Limite d'un mot
- \B pas limite d'un mot
- ^ début de phrase
- $ fin de phrase

Assertions

Lookahead assertions

- (?=...), réussit si la regexp match l'expression à cette position.
- (?!...), l'inverse
- L'assertion laisse le curseur de lecture là où l'on commence a tester l'assertion (c'est à dire que si l'assertion est vraie on va relire depuis le début du groupe assertif)

Exemple

- Matcher les fichiers et leurs extensions
- `.*[.].*$`
- Mais sans les .bat

Solutions

Ce qui ne marche pas

- `.*[.][^b].*$`
- `.*[.]([^b]..|.[^a].|..[^t])$`
- `.*[.]([^b].?.?|.[^a]?.?|..?[^t]?)$`

Solution avec lookahead

- `.*[.](?!bat$).*$`
- Si après le . il n'y a pas bat, on continue.
- Le .* après l'assertion matchera l'extension (la lecture reste bloquée au début de l'assertion/)

Solutions

Un pas en avant, trois pas en arrière !

- (?<=...), réussit si la regexp match l'expression en arrière de cette position.
- (?<!...), réussit si ne match pas (en arrière)

Masques conditionnels

Syntaxe

- `(?(assertion ou backreference) motif si vrai | motif sur faux)`

Exemple

- Si je détecte From dans un mail, je traite un motif d'adresse email
- Si je détecte Subject, je traite une suite de mots

Failing attempts

Capture vide, et groupe non participant

- `(q?)b\1` macth b
- q ? macth le vide, le groupe capture vide, b match b, 1 doit matcher vide c'est ok.
- `(q)?b\1` ne match pas b
- (q) match le vide (optionnel), le groupe ne capture rien, car il ne participe pas à l'expression. b match b, `\1` va échouer car le groupe n'a pas participé.

Répetitions

Que se passe-t-il pour la capture si

- J'utilise `azerty([0-9])+`
- La capture ([0-9]) risque d'intervenir plusieurs fois.
- `\1` ne se souviendra que de la dernière capture.

Forward reference

Utilisons une capture avant de la définir

- Fonctionne avec PCRE
- `(\2deux|(un))+` va matcher : ununduex
- Au début, \2 échoue a matcher u, le seconde alternative "un" réussi (un est capturé)
- Deuxième itération \2 match "un" et deux match deux.

Quelques options

PCRE

- On précise l'option après le # final.
- i : case insensitive
- s : . macth aussi le retour à la ligne
- U : ungreedy par défaut
- x : permet d'écrire ses motifs sur plusieurs lignes, avec des commentaires (#, il faut utiliser un autre délimiteur (ex. /))