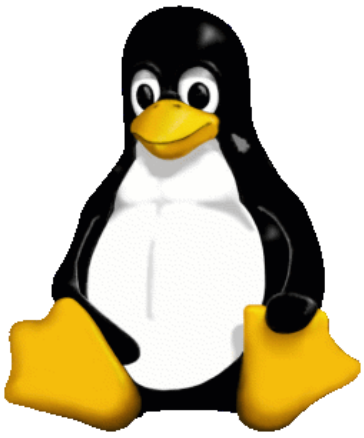
http://vader-fr.fr/spip.php?article173

Expressions régulières

- Linux & Logiciels Libres -



Publication date: mercredi 27 juillet 2016

Que ce soit sous Linux Bash (sed, grep, find...), PHP (preg_match et preg_replace) ou Perl, il est possible de simplifier des traitements en faisant correspondre une chaîne à un motif (pattern).

Certains motifs utilisent des expressions régulières étendues, nécessitant une option (-r pour sed, -E pour grep...) ou un paramétrage à vérifier (PHP/Perl).

Article non exhaustif

Les caractères spéciaux (^, $, ?, +, *, [,], (,)...) seront protégés par un `\` afin de ne pas être interprétés

Généralités

Les composantes d'une expression régulière sont :

- les ancres - elles définissent quelle partie de la chaîne il faut comparer au motif.
  - `^` pour le début de la chaîne
  - `$` pour la fin de la chaîne.
  - en cas de comparaison (et/ou remplacement) d'une partie de chaîne, l'option `g` permettra un traitement pour chaque correspondance au motif.

- les méta-caractères
  - `.` signifie "n'importe quel caractère". Un "." simple devra être protégé par un `\`

- les types de caractères, indiqués entre `[ ]` - valable pour un seul caractère à comparer. Il faudra utiliser des quantificateurs pour comparer plusieurs caractères.
  - au début de `[ ]`, le `^` indique une négation. Ainsi, `[^0-9]` signifie "tout sauf numérique"
  - `[0-9]` pour du numérique.
  - `[a-z]` pour de l'alphabétique en minuscule
  - `[A-Z]` pour de l'alphabétique en majuscule
  - `[a-zA-Z0-9]` pour de l'alphanumérique.
  - Il est possible de lister les valeurs `[abcdef567]` ou de mettre une liste implicite `[a-f5-7]`
  - le "-" en tant que caractère sera donc mis en premier dans la liste, afin de ne pas tester une liste implicite. exemple, soit "-", soit 0, soit 9 : `[-09]`
  - il existe aussi des types définis et des raccourcis
    - `\d` est ainsi l'équivalent de `[0-9]` et `\w` celui de `[a-zA-Z0-9]`, `\s` un caractère d'espacement (espace, tabulation...), `\t` une tabulation simple, `\b` indique un séparateur de mot...
    - les inverses sont alors en majuscule. `\D` étant tout sauf numérique.

  - le respect ou non de la casse pourra être indiqué par l'option `i`, juste en dehors de l'expression régulière

- les sous-chaînes, entre `( )` - permettent de comparer une .. sous-chaîne, pouvant être composée de plusieurs expressions
  - exemple, `^(Q|q)$` correspond à un q majuscule ou minuscule, le "ou" logique étant représenté par `|`. C'est l'équivalent de `^[qQ]$`
  - `^(perl|linux|php)$` à la chaîne entière "perl" OU "linux" ou "php". contrairement à la notation `[ ]`, la comparaison se fait sur plusieurs caractères.

- une chaîne unique, sans utilisation de OU logique ou de répétition de la sous-chaîne, pourra être simplement indiquée. voir exemple qui suit.
- autre exemple `^Mem(Free|Total):` cherche en début de chaîne "MemFree :" ou "MemTotal :"
- exemple dans ce code PHP `$patterns["max_essais_logins"]="/^[a]{0}(0*[3-9]|(0*[1-9]0*)+0|(0*[1-9]+0*)+[1-9])$/";` qui contrôle une chaîne numérique supérieure à 3.
  - le `[a]{0}` sépare le `^` de la sous-chaîne, afin de le comprendre comme "début de chaîne globale" et non "différent de sous-chaîne qui suit".

- Il n'existe pas de "et" logique en expression régulière, mais il est possible de le simuler par un non "ou" des valeurs inverses.
  - le "non" logique se met devant l'ouverture de la sous-chaîne, exemple `^(valeur1|valeur2)`

- Les sous-chaînes sont automatiquement mises dans des variables numérotées à partir de 1 et peuvent être réutilisées dans un remplacement.
  - exemple `s/^.+: +([0-9]+) kB/$1/` via un remplacement, extrait de la chaîne les seuls caractères numériques situés avant un " kB", et ce pour la première occurrence dans la ligne.

- les quantificateurs - situés après le type de caractère ou la sous-chaîne. Par défaut, vaut 1.
  - `?` signifie "zéro ou 1 fois"
  - `*` signifie "zéro à n fois" (sans limite)
  - `+` signifie "au moins 1 à n fois"
  - `{a,b}` signifie "de a à b fois". a ou b peuvent être absents, afin de ne spécifier que le maximum ou minimum.
    - ainsi, `{0,}` est un équivalent de `*`
    - exemple `^(\.|\.\.)$`, afin de ne pas traiter "." et ".." dans la liste des répertoires Unix/Linux, peut être remplacé par `^\.{1,2}$` ou `^\.\.?$`

Exemples d'utilisation

- PHP
  - preg_match permet de comparer le contenu d'une variable à un motif. preg_replace fera le remplacement.
  - Ici, on détermine le type de donnée avant un bindValue d'une requête paramétrable (PDO)
  - les caractères encadrant le motif étant `/`, les options s'indiqueraient de manière "/motif/options"
    - par exemple `preg_match("/^linux/i",$chaine)` sera insensible à la casse et cherchera en début de chaîne le terme "linux" avec ou sans majuscules.

  - `if (preg_match("/[\w]+/",$param)){ $typ_param=PDO::PARAM_STR; }` si la chaîne contient (peut importe où) un voire plus caractères alphabétiques, c'est une chaîne de caractères
    - `\w` peut être remplacé par `\D`, signifiant "tout sauf numérique". Ici, l'expression est assez permissive car c'est le type "par défaut", qui peut être surchargé par les deux suivants.

  - `if (preg_match("/^[+-]?\d+$/",$param)){ $typ_param=PDO::PARAM_INT; }` si la chaîne ne contient, du début à la fin, qu'un (ou pas) +/- suivi d'un ou plusieurs caractères numériques, c'est un entier signé
  - `if (preg_match("/^$/",$param)){ $typ_param=PDO::PARAM_NULL; }` si la chaîne, entre son début et sa fin,

ne contient rien, c'est un paramètre null
- Le motif suivant vérifie si la chaîne entrée est un numérique, supérieur ou égal à 3
- `$patterns["max_essais_logins"]="/^[a]{0}(0*[3-9]|(0*[1-9]0*)+0|(0*[1-9]+0*)+[1-9])$/";`
  - commence par zéro fois la lettre a. En fait, cela permet d'interpréter le `^` comme "début de chaîne" et non "non égal à la sous-chaîne qui suit"
  - puis contient ensuite soit `0*[3-9]` soit `(0*[1-9]0*)+0` soit `(0*[1-9]+0*)+[1-9]` répété au moins une fois, et termine
  - `0*[3-9]` = un chiffre de 3 à 9 tout seul, précédé ou non de zéros.
  - `(0*[1-9]0*)+0` = une chaîne contenant au moins un chiffre de 1 à neuf, et qui terminera par zéro
  - `(0*[1-9]+0*)+[1-9]` = une chaîne contenant au moins un chiffre de 1 à neuf, et qui terminera par un chiffre de 1 à neuf, donc complémentaire à la chaîne précédente

- PERL
  - `=~` et `!~` permettent de comparer le contenu d'une variable à un motif.
  - Ici, on utilise un opérateur ternaire : si la valeur commence éventuellement par + ou -, suivi d'un ou plusieurs chiffres, on met la valeur dans le tableau, sinon on sort.
    - l'opérateur ternaire existe aussi en PHP. exemple : `$page=(isset($_POST["page"]))?$_POST["page"]:"accueil";` affecte à la variable page la valeur transmise en POST si celle-ci existe, sinon la valeur "accueil".

  - `($valeur=~/^[+-]{0,1}[0-9]+$/)?push(@rray,$valeur):last;`
  - `s/<\/{0,1}head>//gi;` le s indique que l'on remplace (substitute) la chaîne ou (le / étant protégé), par une chaîne vide, avec les options g (global, ne pas s'arrêter à la première occurrence) et i (insensible à la casse)

- Linux : find et sed, voir ici
- Autres
  - `3\.14(15?)?` 3.14 ET 3.141 ET 3.1415
  - `(ab)([c]{0,1}|[c]{3})z` abz ET abcz ET abcccz
  - `(abc)[c]{2}z` abcz ET abcccz MAIS PAS abz
  - `(ab)[c]{0,1}z` abz ET abcz
  - `[\\]*[\*]*` un nombre quelconque de "\" suivi d'un nombre quelconque de "*"
  - `([^"]*)` une chaîne entre quotes contenant 0 à n fois tout sauf "
  - `^0?[0-3]?[0-7]{1,2}$` commence par 0 (0 ou 1 fois) puis 0 à 3 (0 ou 1 fois) puis 0 à 7 (1 ou 2 fois) et termine
  - `\b[\w.]{1,12}\b` un mot de 1 à 12 caractères alphanumériques ou "."
  - `^\$\w+$` une chaîne contenant uniquement le nom d'une variable scalaire sans toutefois correspondre à une variable spéciale comme `$*` (PERL)
  - `^(#.*|\s*)$` une chaîne commençant par soit un dièse suivi de n'importe quoi en n'importe quelle quantité soit ne contenant (éventuellement) que des caractères de séparation, jusqu'à la fin de la chaîne. un `\s*#` au lieu du simple `#` permettrait de gérer les lignes de commentaires commençant par des caractères d'espacement.