11 Expressions régulières

La mise en concordance des expressions régulières (regex) est un moyen puissant de rechercher des séquences qui répondent à un certain modèle, par exemple tout mot qui consiste uniquement en lettres capitales ou toutes les occurrences de deux mots identiques séparés par une ou plusieurs espaces, ou d'innombrables autres modèles.

L'expression régulière la plus simple est juste une séquence à rechercher. Une regex du genre

```
xyz
```

recherche simplement la séquence " xyz ". Toutefois, il y a quelques caractères qui ont une signification spéciale dans les expressions régulières. Ces caractères sont :

```
( ) [ ] ^ $ . * + ? { } | \
```

Si vous voulez rechercher une occurrence littérale d'un de ces caractères, vous devez le faire précéder d'une barre oblique inversée. Si, par exemple, vous voulez rechercher la séquence " xyz? " vous devez utiliser l'expression régulière

```
xyz\?
```

Pour rechercher une barre oblique inversée, utilisez \\. Une exception à cette règle c'est lorsque les caractères spéciaux se produisent à l'intérieur d'une gamme (c'est-à-dire entre les crochets, voir ci-dessous), alors ils n'ont pas besoin d'être précédés d'une barre oblique inversée.

Bon, alors que font donc ces caractères spéciaux ? Un point (.) est un atout qui correspond à tout caractère (un exactement). Par exemple, l'expression régulière

```
d.g
```

coïncidera avec les séquences " dog ", " dig ", " d=g " etc. Elle ne recherchera cependant pas la séquence " dg ".

Les caractères * et + sont des opérateurs de répétition, ils recherchent des occurrences multiples du modèle précédent. Le * recherche des occurrences zéro ou plus, le + une ou plus. Par exemple, l'expression régulière

```
a+
```

recherchera les séquences " a ", " aa ", " aaaaaa " etc. Normalement ces opérateurs ne s'appliqueront qu'au seul caractère qui les précède. Par exemple la regex

```
ab*c
```

recherchera les séquences " ac ", " abc ", " abbbc " etc, mais non la séquence " ababc ". Si vous voulez que ces opérateurs s'appliquent à plus d'un caractère qui précède, vous devez utiliser des parenthèses. Par exemple,

```
a(xy)*b
```

recherchera les séquences " ab ", " axyb ", " axyxyb " etc.

Le point d'interrogation fonctionne de manière similaire à * et +, mais il cherche zéro ou une occurrence du modèle précédent. Par exemple,

```
ab?c
```

recherchera les séquences " ac " et " abc " et rien d'autre.

La barre verticale (|) est un opérateur OU. Elle cherche les occurrences de son opérande, soit droite, soit gauche. Par exemple, l'expression

```
abc|xyz
```

cherchera les séquences " abc " et " xyz " et rien d'autre. À la différence des opérateurs *, +, et ?, l'opérateur | ne s'applique pas uniquement aux seuls caractères qui l'entourent, mais à tout jusqu'y compris le début et la fin de l'expression régulière. Si vous voulez le limiter à un sous-ensemble au milieu d'une regex, vous devez utiliser des parenthèses. Par exemple,

```
d(i|ra)g
```

coïncidera avec la séquence " dig " ou " drag ".

^ et $ limitent respectivement une recherche au début et à la fin d'une ligne. Par exemple,

```
^xyz
```

ne recherchera la séquence " xyz " que si elle est au début d'une ligne, et

```
xyz$
```

ne recherchera la séquence " xyz " que si elle est à la fin d'une ligne. L'expression régulière

```
^xyz$
```

recherchera toute ligne qui ne contient que la séquence " xyz ".

Les crochets peuvent être utilisés pour spécifier un ensemble de caractères. Par exemple,

```
[abcd]
```

coïncidera avec les seuls caractères " a ", " b ", " c " ou " d " (rien d'autre). Vous pouvez utiliser un trait d'union pour spécifier une plage de caractères, comme

```
[0-9]
```

pour chercher un seul chiffre. Vous pouvez combiner des plages et de simples énumérations de caractères ; par exemple,

```
[a-zA-Z0-9+=]
```

coïncidera avec une lettre en haut ou bas de casse, un nombre, ou un des caractères + ou =. Si vous voulez inclure un tiret dans l'ensemble, il doit être au début ou à la fin (autrement il sera traité comme un opérateur de plage).

Si le premier caractère après le crochet ouvrant est un ^, la regex assortira tout caractère qui n'est pas contenu dans l'ensemble.

{ } est un autre opérateur de répétition, similaire à * et + excepté que vous pouvez spécifier le nombre minimum et maximum de répétitions. La syntaxe élémentaire est `p{m,n}` où p est le modèle à rechercher (soit un caractère simple ou un sous-modèle dans des parenthèses), m est le nombre minimum et n le nombre maximum de répétitions. Par exemple,

```
(xy){2,4}
```

recherchera les séquences “xyxy ”, “ xyxyxy ”ou “ xyxyxyxy ” (rien d’autre).

Les formes spéciales sont `{n,}` pour chercher n répétitions ou plus et `{n}` pour chercher exactement n répétitions. Par exemple,

```
x{5}
```

recherchera la séquence “ xxxxx ” et rien d’autre.

La barre oblique inversée est toujours utilisée en combinaison avec le caractère qui le suit. Nous avons déjà vu qu’elle peut être utilisée pour “ échapper ” des caractères spéciaux lorsque vous voulez rechercher des occurrences littérales de l’un deux (par exemple, `\*` recherche *). D’autres utilisations de la barre oblique inversée : `\t` recherche un caractère de tabulation, et `\n` recherche un retour chariot. `\<` limite la recherche au début d’un mot et `\>` à la fin d’un mot. Par exemple,

```
\<A[a-z]*
```

recherchera tout mot qui commence par A majuscule, et

```
\<Mac\>
```

ne recherchera que le mot “ Mac ” mais pas “ Macintosh ” ou “ MacSOUP ”.

Examinons maintenant quelques exemples instructifs :

1)

Supposez que vous voulez créer un filtre de classeur pour quelqu’un qui rédige du courrier électronique à partir de plusieurs machines différentes sur son réseau, afin que son en-tête From puisse être “ joe@machine1.host.domain ” ou “ joe@some_other_machine.host.domain ” etc. Vous pourriez, évidemment, créer des filtres distincts pour toutes ces adresses si vous les connaissez toutes, mais vous pouvez également utiliser la regex

```
joe@.+\.host\.domain
```

Rappelez-vous que vous devez “ échapper ” les points littéraux avec une barre oblique inversée.

2)

Supposez que vous voulez éliminer tous les articles d’une longueur supérieure à 1 000 lignes. Vous pourriez créer une entrée de filtre pour la ligne d’en-tête Lines et utiliser une regex du type

```
[0-9]{4,}
```

Celle-ci coïncidera avec tout nombre consistant en quatre chiffres ou plus. L’élimination d’articles de, disons, plus de 300 lignes, est un peu plus compliquée. Vous ne pouvez pas utiliser la regex

```
[3-9][0-9]{2,}
```

parce que celle-ci manquera les nombres 1 000 à 2 999 et 10 000 jusqu’à 29 999 etc. Vous pourriez utiliser

```
[3-9][0-9][0-9]|[0-9]{4,}
```

à la place. La première moitié de cette expression coïncide avec les nombres 300 jusqu'à 999, et la deuxième moitié avec 1 000 et au-delà.

Une bonne manière de vérifier des expressions régulières pour la ligne d'en-tête Lignes est d'ouvrir un groupe de Nouvelles, trier par Lignes, désactiver " Regrouper les enfilades " et d'essayer ensuite l'expression régulière avec la commande Rechercher.

Soyez prudent lorsque vous mettez pareille entrée de filtre dans votre ensemble <Tous les groupes> ! L'auteur l'a fait une fois et après un moment, a commencé à se demander pourquoi plus rien n'apparaissait dans comp.sys.mac.digest…

3)

Supposez que vous vouliez éliminer tous les articles qui sont transpostés dans 5 groupes ou plus. Créez une entrée de filtre pour la ligne d'en-tête Newsgroups et utilisez l'expression régulière

```
([^,]+,){4,}
```

La sous-expression entre parenthèses recherche une séquence d'un ou plusieurs caractères autres que la virgule (c'est-à-dire un nom de groupe de Nouvelles), suivie d'une virgule. Cette sous-expression doit coïncider quatre fois ou plus, c'est-à-dire que la ligne d'en-tête Newsgroups doit contenir quatre virgules ou plus afin que la regex coïncide. Si vous avez suivi fidèlement, vous remarquerez que cette regex apparie aussi les lignes d'en-tête Newsgroups qui se terminent par une virgule, mais ne contiennent que quatre groupes. Comme pareille ligne d'en-tête est invalide et ne peut être créée que par un lecteur de Nouvelles défectueux, cette regex fait généralement ce que nous voulons.

Note : il est souvent possible d'écrire plusieurs expressions régulières différentes qui font la même chose. Par exemple, la regex ci-dessus pour rechercher des articles transpostés pourrait également être écrite sous la forme

```
(.+,){4,}
```

Toutefois, si vous essayez les deux expressions dans la zone de dialogue Rechercher, vous verrez que la seconde est beaucoup plus lente que la première. En général vous devriez toujours essayer une regex dans la zone de dialogue Rechercher, avant de l'utiliser dans une entrée de filtre d'élimination, primo pour voir si elle fait réellement ce que vous voulez (il est si facile de faire des erreurs) et secundo pour vous assurer que ça ne dure pas une éternité.