R1.04 - Systèmes - TP 6
Regex & Grep

Expressions Rationnelles

Définition

Une *Expression Rationnelle* est une formulation permettant de décrire, à l'aide d'**une chaîne modèle unique**, un ensemble de chaînes de caractères.

Nommage

L'appellation officielle française est *Expression Rationnelle*.

En anglais on appelle cela une *Regular Expression* ou, sous forme abrégée : *Regex*.

De l'appellation anglaise est née l'anglicisme *Expression Régulière*, et c'est très souvent sous cette appellation qu'on la trouve désormais dans la littérature informatique, et bien évidemment dans le langage courant des informaticien.ne.s.

Nous utiliserons donc aussi cette appellation dans ce TP, et même plutôt le terme de *Regex*.

Exemple

On a déjà rencontré un exemple de mécanisme de Regex rudimentaire : les *Jokers* du Shell.

Ecrire **`*.c`** pour représenter les trois fichiers **`exo1.c`**, **`exo2.c`** et **`demo.c`**, présents dans le répertoire courant est une façon de décrire, à l'aide d'une unique chaîne, un ensemble de 3 chaînes (ici 3 fichiers). C'est une Regex.

Terminologie

Motif

Une Regex s'appelle aussi un **motif** ou **pattern** en anglais.

Le motif est une chaîne et ce qu'il représente peut être : rien, une simple chaîne ou de multiples chaînes. C'est le contexte qui déterminera ce que le motif représente réellement. Pour les Jokers, le contexte est donné par le contenu du morceau d'arborescence visé. Parfois un **`*.c`** représente 3 fichiers et parfois rien du tout.

Il existe plusieurs notations pour définir les règles de description des motifs. On vient de dire que les Jokers en sont une. Nous allons voir une autre notation dans ce TP (voir *Outils* ci-après), et nous nous limiterons à celle-là car elle est déjà très répandue.

Correspondance

Quand un motif trouve une ou plusieurs **correspondances**, on appelle cela **matcher**[1]. C'est un anglicisme courant, et on va aussi l'utiliser car il trouve écho dans le nom de fonctions Regex (en C, en PHP, en JS, en Java, et dans bien d'autres langages) souvent nommées **`match(...)`**.

Caractère

Ne pas confondre un caractère et une lettre. Une lettre est un caractère, un caractère n'est pas forcément une lettre. Par exemple, les chiffres décimaux sont aussi des caractères.

Un caractère est une des cases de la table ASCII (https://www.asciitable.com). Un caractère est souvent affichable, mais il y a certains caractères qui n'ont pas de symbole graphique associé. Ils n'en demeurent pas moins des caractères. Par exemple, le code **`ASCII 7`** = **`bell`**, émet un son dans le Terminal au lieu d'afficher quelque chose. Il occupera pourtant un octet dans un fichier.

Dans les exemples et les questions suivantes, on parlera généralement de caractères.

Outils

Les Regex sont un mécanisme très utile en informatique. De nombreux langages et librairies existent pour mettre en œuvre ce mécanisme.

Sous Linux, il existe une commande qui est associée aux Regex : **`grep`** (**G**lobally search **R**egular **E**xpression and **P**rint[2]).

Nous utiliserons une version améliorée du **`grep`** de base. Cette version s'appelle **`egrep`**[3].

Par habitude (et abus) de langage, on dit simplement **`grep`** mais on utilisera plutôt la commande **`egrep`**.

[1] De l'anglais : to match (correspondre)
[2] Cherche globalement un Regex et affiche le résultat
[3] Extended grep

C'est un outil extrêmement puissant et utile, qu'il convient de connaître et maîtriser. Vous ~~pouvez~~ devez en (ab)user.

La commande egrep

On va étudier les Regex en utilisant **egrep**.

Les Regex ne sont pas **egrep**.

egrep utilise des Regex.

Fonctionnement

egrep recherche la présence d'un motif dans une source de données texte. On peut aussi l'utiliser sur des fichiers binaires, même si c'est un usage plus rare et qui nécessite des options spéciales (voir son **man** si vous en avez besoin).

Par défaut, **egrep** affiche[4], sans altération, chaque ligne de la source qui correspond au motif, les autres lignes sont simplement ignorées[5].

egrep peut être utilisé comme une commande autonome :

```
egrep motif fichier_de_données
```

ou encore :

```
egrep motif < fichier_de_données
```

Mais **egrep** est aussi un filtre et peut donc s'utiliser ainsi :

```
une_commande ... | egrep motif
```

et peut bien évidemment servir aussi, comme n'importe quel filtre, à alimenter d'autres filtres à sa suite :

```
une_commande ... | egrep motif | une_autre_commande ...
```

[4] Plus précisément, **egrep** envoie vers son **STDOUT**
[5] Elles ne sont donc pas envoyées vers le **STDOUT** de **egrep**.

Syntaxe

```
egrep [options...] motif
```

On va étudier en détail la partie **motif**, juste après.

Les options sont multiples, une consultation du **man** donnera beaucoup d'informations sur le sujet mais voici déjà quelques options que nous allons utiliser fréquemment :

- **-i** : ne fait pas de différence entre minuscules et majuscules (**i**gnore case)
- **-l**[6] : n'affiche pas les lignes qui matchent mais affiche juste les noms des fichiers dans lesquels **egrep** a trouvé au moins une ligne qui matche le motif.
- **-v** : inversion/négation du match. Affiche les lignes qui ne matchent pas (in**v**ert)
- **-c** : affiche simplement le nombre de lignes qui ont matché le motif (**c**ount)

Motifs

ATTENTION, ne confondez pas les Jokers du Shell et les Regex de egrep !

Il y a des choses identiques, des choses similaires et plein de choses totalement différentes. Prenez des notes, ne mélangez pas les deux.

Utilisez les fichiers du *TP 5 (suite)* ainsi que le fichier **quant** pour tester les exemples qui vont suivre.

Notez vos résultats sur une feuille ou dans un document texte.

Vérifiez bien que votre affichage correspond à votre motif et que les lignes qui ne doivent pas matcher ne s'affichent effectivement pas : vous avez une astuce pour colorer les matchs un peu plus loin, utilisez-la à bon escient !

L'objectif n'est pas que quelque chose s'affiche mais bien que s'affichent les bonnes choses et uniquement celles-là !

Chaîne simple

C'est la forme la plus élémentaire : la recherche d'une simple chaîne composée exactement des caractères indiqués dans le motif.

Exemple de recherche de la chaîne *machine* dans le fichier **lang** :

[6] **L** minuscule

```
egrep machine < lang
```

Q.1 Affichez les lignes qui contiennent la chaîne **blanche** dans le fichier **prod**.

Q.2 Affichez les lignes qui contiennent la chaîne **blan** dans le fichier **prod**. Avez-vous le même résultat ?

Conclusion : il n'est pas nécessaire de rechercher des mots complets. On peut parfois se satisfaire de parties de mots.

Q.3 Affichez les lignes qui contiennent la chaîne **lan** dans le fichier **prod**. Avez-vous le même résultat ? Pourquoi ?

Afin de mieux visualiser les parties qui matchent le motif, vous pouvez maintenant taper ceci dans votre Terminal :

```
alias egrep='egrep --color'
```

Puis, refaites maintenant un essai avec l'exemple de la Q.3 pour voir le changement.

Désormais, et jusqu'à la fermeture du Terminal, vous bénéficiez de la coloration des matchs du motif. Mais, si vous fermez votre Terminal, vous devrez relancer la commande **alias** pour retrouver cet effet.

Liste

A l'instar des listes pour les *Jokers* du *Shell*, on peut aussi spécifier des listes de caractères en Regex. La syntaxe est exactement la même. Une liste représente la présence d'un et un seul caractère parmi l'ensemble listé.

Rappels :

- Une liste peut s'exprimer in extenso : **[ABCDE]** ou par intervalle : **[A-E]**
- Une liste peut combiner plusieurs écritures : **[A-Za-z]** pour toutes les majuscules et toutes les minuscules (52 lettres au total) ou encore **[ABC-FYZ]** pour **[ABCDEFYZ]** (une lettre parmi les 8 de la liste).
- Pour faire figurer un **-** (tiret), il suffit de le mettre en 1er ou en dernier pour éviter qu'il ne soit considéré comme un intervalle : **[ABC-]** (4 caractères)

Exemple de recherche d'un chiffre dans le fichier **lang** :

```
egrep [0-9] < lang
```

Si vous avez bien tapé la commande **alias** précédemment, on observe que chaque chiffre du fichier matche le motif. Même si on cherchait la présence d'un seul chiffre, le motif a donc matché un chiffre à plusieurs endroits, y compris sur une même ligne.

Q.4 Affichez les lignes de **prod** qui contiennent un chiffre entre **7** et **9**.

Q.5 Affichez les lignes de **prod** qui NE contiennent PAS un chiffre entre **7** et **9** (piège, vérifiez bien qu'aucun **7**, **8** ou **9** ne s'affiche !).

Q.6 Affichez les lignes de **prod** qui contiennent une voyelle.

Q.7 Affichez les lignes de **prod** qui contiennent consécutivement deux voyelles quelconques.

Q.8 Affichez les lignes de **prod** qui NE contiennent PAS consécutivement deux voyelles quelconques. Vous devez avoir réussi la Q.5 pour réussir celle-ci. Si ce n'est pas le cas, appelez votre enseignant.e.

Q.9 Affichez les lignes de **depts** qui contiennent au moins soit un **A** majuscule, soit un **-** (tiret), soit un **Y** majuscule. (piège)

Un caractère quelconque

En *Joker Shell*, un caractère quelconque est représenté par un **?** (point d'interrogation)

En Regex ce n'est pas du tout pareil. Mettez-le aussi en gros souligné dans vos notes !

Un caractère quelconque s'écrit **.** (point), car le **?** sert à autre chose (vu plus loin).

Exemple de recherche d'un caractère quelconque, encadré de **:** (deux-points), dans le fichier **prod** :

```
egrep :.: < prod
```

Q.10 Affichez les lignes de **depts** qui contiennent un **9** suivi de 3 caractères quelconques, suivis d'un **C** majuscule.

Q.11 Affichez les lignes de **murphy** qui contiennent un **a** suivi d'un **e** deux positions plus loin. (Piège, il en a 5 seulement)

Pour faire figurer le caractère **.** (point) dans le motif, il suffit de le préfixer d'un **** (backslash). On en reparlera à la fin de ce TP.

Q.12 Affichez les lignes de **murphy** qui contiennent **...** (3 points de suspension). Faites-vous aider si ça ne marche pas bien.

Espace

Pour le moment, dans les exemples précédents, nous avons écrit les motifs sans utiliser de **"** (guillemet) ni de **'** (apostrophe).

Si vous avez la présence d'un espace dans votre motif, vous devez impérativement placer votre motif entre **"** ou **'**. Si vous n'avez pas d'espace dans le motif, vous pouvez quand même utiliser des **"** ou des **'** si vous le souhaitez. C'est même sans doute une bonne habitude à prendre).

Un espace peut aussi être utilisé sans **"** ou **'**, en le préfixant d'un **** (backslash)

Q.13 Affichez les lignes de **lang** qui contiennent **machine virtuelle**

Quantificateurs

1. Définition

Un quantificateur… quantifie (hé oui !)… ce qui le précède.

Quantifier signifie indiquer quel nombre de fois *le truc qui précède* le quantificateur peut apparaître.

Pour le moment, ce qui précède = le caractère qui précède.

Dans : **toto<quantificateur>**, ce qui précède le *quantificateur* n'est pas **toto** mais le dernier **o**.

Dans : **[A-Z][0-9]<quantificateur>**, ce qui précède le quantificateur n'est pas une lettre majuscule suivie d'un chiffre, mais uniquement le chiffre.

On peut appliquer un quantificateur sur un groupe de caractères : voir le paragraphe sur **Groupage**, plus loin.

2. Une quantité quelconque

En *Joker Shell*, le caractère ***** (étoile) signifie *un nombre quelconque de caractères, y compris aucun*.

En Regex ça y ressemble, MAIS ce n'est pas pareil. Mettez-le aussi en gros souligné dans vos notes !

Le caractère ***** est un **quantificateur**, qui signifie bien, comme pour les *Jokers Shell* : une quantité quelconque. Mais, alors qu'en Joker Shell c'était une quantité quelconque de n'importe quels caractères, en Regex c'est une quantité quelconque du truc qui est avant le *****

Exemple :

```
egrep A[0-9]*B < quant
```

signifie un **A**, suivi d'une quantité quelconque (y compris nulle) de chiffres, suivie d'un **B**.

Ca matche donc **A5B**, **A123B**, **A999B** et même simplement **AB**.

Un quantificateur ne signifie pas qu'il doive y avoir répétition du même *truc*.

Q.14 Affichez les lignes de **prod** qui contiennent un **b** suivi d'un nombre quelconque de minuscules, suivi d'un **e**

Q.15 Même question que Q.14 mais le **e** doit être suivi d'un **:** (deux-points)

Q.16 Saurez-vous afficher les lignes de **depts** qui contiennent au moins deux **-** (tirets) dans leur nom ?

3. Au moins une fois

C'est presque comme le *****

Le caractère **+** est le **quantificateur** qui signifie une fois ou plus le *truc* qui est avant le **+**

Exemple :

```
egrep A[0-9]+B < quant
```

signifie un **A**, suivi d'une quantité quelconque (non nulle) de chiffres, suivie d'un **B**.

Comme pour *****, ça matche **A5B**, **A123B**, **A999B** mais cette fois-ci, ça ne matche pas **AB**.

Q.17 Affichez les lignes de **depts** qui contiennent une voyelle suivie d'au moins un **l**[7] suivi d'un **e**. (4 lignes).

[7] **L** minuscule

4. Zéro ou une fois

En *Joker Shell*, le caractère **?** (point d'interrogation) signifie *un caractère quelconque et un seul*.

En Regex ce n'est pas du tout pareil. Mettez-le aussi en gros souligné dans vos notes !

Le caractère **?** est un **quantificateur**, qui signifie la présence de zéro ou une fois le *truc* qui est avant le **?**

On peut aussi dire : le truc avant le **?** peut être présent mais ce n'est pas une obligation.

Exemple :

```
egrep AB?C < quant
```

signifie un **A**, suivi ou pas d'un et un seul **B**, suivi d'un **C**.

Ca matche donc **ABC** et **AC** mais pas **ABBC**.

Q.18 Affichez les lignes de **lang** qui contiennent un **9** suivi éventuellement d'un **7** et suivi (obligatoirement) d'un **0**

Q.19 Affichez les lignes de **murphy** qui contiennent un mot finissant par un **t**. Aide : un mot est suivi d'un espace, mais il y a parfois une virgule avant l'espace. Il y a 12 lignes qui matchent.

5. Exactement X fois

L'expression **{X}** est un **quantificateur** qui signifie la présence d'exactement **X** fois le *truc* qui est avant.

Exemple :

```
egrep AB{3}C < quant
```

signifie un **A**, suivi d'exactement 3 **B**, suivi d'un **C**.

Ca matche donc **ABBBC** et **ZABBBCDE** mais pas **ABBC** ni **ABC**.

Attention, le terme *exactement* peut être trompeur. Il signifie bien exactement s'il est suivi d'autre chose, mais s'il termine le motif, rien n'impose qu'il n'y ait pas plus que **X** fois le truc d'avant. Exemple :

```
egrep AB{3} < quant
```

matche **ABBB** mais matche aussi **ABBBBBB** car on ne précise rien derrière l'obligation d'avoir **3** fois la lettre **B**. Derrière il peut y avoir n'importe quoi, y compris encore un ou plusieurs **B**.

Q.20 Affichez les lignes de **depts** qui contiennent exactement 2 **r** consécutifs, suivis d'un **i**.

Q.21 Affichez les lignes de **depts** qui contiennent 2 **n** consécutifs immédiatement après un **e**.

6. Entre X et Y fois

L'expression **{X,Y}** est un **quantificateur** qui signifie la présence de **X** à **Y** fois le *truc* qui est avant.

Exemple :

```
egrep AB{2,3}C < quant
```

Ca matche donc **ABBC** et **ZABBBCDE** mais pas **ABBBBC** ni **ABC**.

Positionneurs

1. Définition

Un positionneur… positionne (hé oui encore !)… l'endroit où le motif est attendu.

2. Début

Pour indiquer que le motif est attendu en début de ligne, on utilise le **^** (circonflex[8])

Exemple :

```
egrep ^A < quant
```

signifie un **A** en début de ligne.

Ca matche donc, des lignes débutant par exemple avec : **ABC** , **AAC** et même simplement **A** mais pas **BA**.

[8] Caret en anglais

Q.22 Affichez les lignes de **murphy** qui commencent par un **L** majuscule.

3. Fin

Pour indiquer que le motif est attendu en fin de ligne, on utilise le **$** (dollar)

Exemple :

```
egrep A$ < quant
```

signifie un **A** en fin de ligne.

Ca matche donc, des lignes terminant par exemple avec : **CBA** , **ZA** et même simplement **A** mais pas **AB**.

Q.23 Affichez les lignes de **murphy** qui se terminent par un **.** (point). Attention au piège.

Cas spécial avec ^

On a vu que le **^** est un positionneur de motif.

Il existe un cas spécial pour ce **^**, lorsqu'il est présent en tête d'une liste. Exemple :

```
egrep [^A-Z] < prod
```

signifie qu'on souhaite le complément de **[A-Z]**, c'est-à-dire tout sauf **[A-Z]**.

Ça revient à écrire une liste de tous les caractères, sauf les lettres majuscules.

ATTENTION, on ne cherche pas l'absence de ces caractères mais bien la présence d'autre chose que ces caractères. La nuance est subtile mais essentielle !

Avec les Regex, on s'intéresse toujours à la présence

de quelque chose, pas à son absence.

Il suffit qu'on ait la présence d'au moins un caractère de la liste (une liste de tous les caractères sauf les majuscules) pour que ça matche, même si on a éventuellement aussi la présence de majuscules !

Chercher l'absence se fait en utilisant l'option **-v**. Ne confondez pas !

Q.24 Affichez les lignes de **depts** qui contiennent autre chose que des minuscules, des majuscules, des chiffres, des espaces et des **:** (deux-points). (41 lignes)

Note (oui, on insiste pour que ça rentre bien) : on cherche les lignes qui contiennent autre chose et non pas les lignes qui ~~NE contiennent QUE autre chose~~.

Groupage

On a parlé principalement de caractères pour le moment. Même si on a effectué des recherches de chaîne, il s'agissait finalement d'une succession de caractères juxtaposés. Exemple :

```
egrep ABB < quant
```

cherche la présence d'un **A** suivi immédiatement d'un **B** suivi immédiatement d'un autre **B**.

Notre esprit globalise cela en : *chercher la suite **ABB** dans **quant***

Mais, quand on a vu les quantificateurs, on s'est rendu compte que **egrep** considère bien ce **ABB** comme 3 caractères individuels et non pas un mot.

En effet, ceci :

```
egrep AB{2} < quant
```

ne signifie pas : *chercher la présence de **2** fois consécutives le mot **AB***, mais *chercher un **A** suivi de **2** fois consécutives un **B***. C'est le **B** qui est quantifié, pas le **AB**.

En utilisant des **()** (parenthèses), on peut opérer un groupage qui permet de résoudre ce problème :

```
egrep (AB){2} < quant
```

Cette fois-ci on cherche la présence de **AB** deux fois consécutives.

Ca matche donc **ABAB** ou encore **ABABABABABABAB**, etc. mais pas **ABB**, ni **AB AB**.

Q.25 Avec un groupage, affichez les lignes de **prod** qui contiennent **an** au moins deux fois de suite.

1. Groupes imbriqués

On peut imbriquer des groupes dans des groupes, de cette façon :

```
egrep (A{3}(XY)+){2} < quant
```

cherche la présence de 3 **A** consécutifs suivis de **XY** une ou plusieurs fois. Tout ceci représente un motif groupé qui doit apparaître 2 fois consécutives.

Ca matche donc : **AAAXYAAAXY** ou **AAAXYXYXYAAAXYXYXY** ou encore **AAAXYXYAAAXYZ**.

Attention, c'est le motif qui est répété 2 fois, pas le texte qui matche le motif. C'est pour cela que dans **AAAXYXYAAAXYZ** la seconde partie (**AAAXY**) est différente de la 1ère partie (**AAAXYXY**).

2. Répétition d'un groupe

On vient de voir qu'un quantificateur appliqué à un groupe quantifie le motif, indique le nombre de fois que le motif doit être ou peut être présent. Il ne concerne pas les données elles-mêmes.

Si on veut indiquer que ce sont les données qui sont répétées, on peut utiliser la syntaxe suivante :

```
egrep ([A-Z]{2})\1\1 < quant
```

cherche la présence d'un groupe de **2** lettres majuscules suivi de ces mêmes 2 lettres, 2 fois de suite : les **\1** faisant référence au texte qui a matché avec le **1**er groupage.

Si plusieurs groupes sont imbriqués, la valeur de **X** dans un **\X** est donnée en comptant le numéro de la parenthèse ouvrante du groupe visé. Exemple :

```
egrep (A{3}(XY)+)\2\1 < quant
```

cherche **3** lettres **A** suivies de **XY** au moins une fois et tout ceci doit être suivi du même texte (**XY** une ou plusieurs fois) trouvé par l'intégralité du **2**ème groupage, suivi du même texte trouvé par l'intégralité du **1**er groupage. C'est une écriture qui peut matcher des textes assez complexes. Il est parfois plus facile d'écrire une Regex que de comprendre ce qu'elle fait, une fois écrite ! Pensez à ceux qui viennent après vous (et vous-mêmes), documentez votre code, merci !

Q.26 Affichez les lignes de **prod** qui contiennent une répétition (une succession) du même groupe de 2 lettres quelconques.

Alternatives

Le quantificateur **?** permet de représenter une alternative : *le truc qui précède peut être présent ou absent*. Cependant, cette alternative est assez limitée.

Il existe une autre solution pour représenter des alternatives, en utilisant le caractère **|** (barre verticale ou pipe).

Attention, il s'agit bien du même caractère que le pipe utilisé pour les tubes, mais ça n'a rien à voir avec les tubes ! Mettez vos motifs entre **"** ou **'** pour éviter que le **|** ne soit interprété comme un tube par le Shell. Exemple :

```
egrep '7[0-9]| m' < lang
```

permet d'afficher les lignes de **lang** qui contiennent soit un **7** suivi d'un chiffre quelconque, soit un espace suivi d'un **m**. Evidemment ça peut être les deux en même temps, mais l'une ou l'autre des parties du motif suffit.

Le motif doit matcher soit la partie gauche, soit la partie droite. Les deux parties sont totalement indépendantes l'une de l'autre. Il est possible d'avoir plus de deux parties dans une alternative. Chaque partie est séparée par un **|**, à gauche, à droite ou même à gauche et à droite.

Q.27 Affichez les lignes de **prod** soit qui commencent par **c** ou **k**, soit qui finissent par **ce**.

Caractères spéciaux

Vous avez remarqué que plusieurs caractères ont une signification spéciale dans l'écriture des motifs : *** ? + [ ] - . { } | ^ $ **. Si vous avez besoin d'utiliser aussi l'un de ces caractères dans la recherche, en tant que ce qu'il représente réellement, il faut l'échapper à l'aide d'un ****. Exemple :

```
egrep '\.' < lang
```

Recherche d'un **.** (point) et non pas d'un caractère quelconque (car **.** signifie *un caractère quelconque* en Regex).