

TP n°1

Utiliser des Expressions Rationnelles en UNIX

Conseils. Les versions des outils utilisés dans ce TP ne sont pas les mêmes sur FreeBSD, Linux ou mac ; pour que les exemples du TP fonctionnent, il vous est donc recommandé d’utiliser les machines du SCRIPT, ou une autre machine sous Linux.

Pendant ce TP, nous travaillerons avec des outils de ligne de commande **Unix**. **Ne pas oublier de garder une trace de son travail en sauvegardant les commandes utilisées** (avec éventuellement de courts commentaires) dans un fichier (une sorte de journal de bord).

1 Les expressions rationnelles étendues de UNIX

Les outils **Unix** utilisent plusieurs versions d’expressions rationnelles. Dans ce TP, nous allons utiliser les *expressions rationnelles étendues* (angl. : *extended regular expressions*). Un petit résumé de ce qu’il est utile de savoir :

1. Symboles : on écrit directement les symboles comme des lettres, des chiffres et les autres symboles quand ils n’ont pas une signification spéciale.
2. Concaténation : elle s’écrit simplement en collant des expressions. Ainsi, `abc` dénote le mot “abc”.
3. Union : c’est le symbole `|` (attention : pas `+`).
4. Parenthèses : comme d’habitude, c’est-à-dire `(,)`.
5. Échappement : le symbole `\` est le symbole d’échappement ; quand il précède un symbole spécial, il le banalise, et inversement il peut donner un sens spécial à certains symboles. Ainsi, `\(` dénote le symbole `(` et `\\` dénote le symbole `\`.
Parmi les lettres qui ont un sens spécial quand elles sont précédées par le symbole d’échappement, il y a en particulier `\w` qui dénote un symbole qui peut faire partie d’un mot : une lettre minuscule ou capitale, un chiffre, ou le symbole `_`. En outre, `\<` dénote le début d’un mot, et `\>` la fin d’un mot.
6. Un symbole quelconque : c’est `.` (un point).
7. Itérations : on écrit les opérateurs d’itération `*` et `+` derrière l’expression à laquelle ils s’appliquent.
Ainsi, `(aa)+b*` dénote une séquence d’un nombre pair non nul de “a” suivi d’un nombre quelconque de “b”.
8. Partie optionnelle : le symbole `?` après une expression e signifie soit la chaîne vide, soit une chaîne dénotée par e . Ainsi, `(aab*)?` dénote soit la chaîne vide, soit un mot qui consiste en deux “a” suivis d’un nombre quelconque de “b”.
9. Itération contrôlée : on peut écrire, pour une expression e et des nombres naturels n et m , l’expression `e{n,m}` qui dénote un nombre de répétitions de e qui est entre n inclus et m inclus. Par exemple, `a{1,3}` dénote un des trois mots “a”, “aa”, et “aaa”. Plus

généralement, $e\{n\}$ est équivalent à $e\{n,n\}$, $e\{,m\}$ est équivalent à $e\{0,m\}$, et $e\{n, \}$ est équivalent à $e\{n, \infty\}$. De même, $e?$ est équivalent à $e\{,1\}$.

10. Les classes de caractères : elles dénotent un ensemble fini de symboles. Une classe s'écrit entre crochets [et], quand le premier symbole après le [est \sim alors on prend son complément. Puis on peut écrire entre les deux crochets les éléments de la classe, sans aucune séparation. Ainsi [abc] dénote une lettre parmi "a", "b", et "c", et [\sim a] est un symbole quelconque *sauf* "a".

On peut aussi écrire entre les crochets certaines expressions qui dénotent une classe de caractères (ces classes peuvent dépendre de l'environnement Linux), en particulier :

- des intervalles comme a-z ; ainsi, [a-z123] dénote un symbole qui est soit une lettre en minuscule, soit un chiffre parmi 1, 2, 3.
- [:lower:] une lettre en minuscule ; ainsi, [AB[:lower:]] dénote A, B ou une lettre en minuscule.
- [:upper:] une lettre capitale.
- [:digit:] un chiffre.

11. \sim est le début d'une ligne, \$ sa fin.

Dans ce TP nous utiliserons l'outil **grep**, déjà utilisé en IS1, avec l'option -E. Cette option permet d'utiliser les expressions rationnelles *étendues* comme expliqué plus haut.

Cet outil lit un flux Unix (par exemple, à partir d'un fichier ou du clavier) ligne par ligne, et cherche dans chaque ligne des occurrences ("*matches*") d'un motif décrit par une expression rationnelle. Un des formats d'un appel de **grep** est le suivant :

```
grep -E [-options] 'motif' [fichier_d'entree] [> fichier_de_sortie]
```

comme **grep -E 'sur' fich1 > fich2** par exemple. Si le fichier d'entrée n'est pas renseigné, **grep** lit l'entrée saisie au clavier (pratique pour tester une commande) ; on termine alors avec CTRL-d. Nous conseillons d'écrire toujours l'expression rationnelle entre apostrophes ' et ' : cela protège les symboles de cette expression rationnelle d'une interprétation par le shell.

Utilisé sans option, **grep** renvoie toutes les lignes qui contiennent le motif. Parmi les options utiles, on peut citer :

- o sort seulement l'occurrence trouvée, pas toute la ligne ;
- n précède la sortie avec le numéro de ligne ;
- x applique le motif à la ligne entière, au lieu de chercher une occurrence dans la ligne ;
- c sort seulement le nombre de lignes trouvées.

On obtient la description complète de cette commande, ainsi que la définition des expressions rationnelles étendues, par la commande **man grep**.

Exercice 1 (Si Proust avait connu les expressions rationnelles)

La page moodle du cours est

<https://moodle.u-paris.fr/course/view.php?id=1638>

S'y inscrire et télécharger le fichier `swann.txt`. On va utiliser `grep -E` pour analyser ce texte.

1. Trouver toutes les lignes du texte qui mentionnent le prénom Odette.
2. Combien de lignes du texte mentionnent le prénom Odette ?
3. Combien de lignes du texte mentionnent le prénom Gilberte ?
4. Combien de fois le texte mentionne-t-il le prénom Gilberte ? Indication : utiliser la bonne invocation de `grep -E` pour trouver toutes les occurrences d'un motif dans un texte, suivi de `| wc -l`
5. Comment expliquer la différence entre les valeurs trouvées aux questions 3 et 4 ? Trouver les lignes du texte qui sont responsables de cette différence.
6. Combien de lignes de texte mentionnent le prénom Gilberte ou le prénom Odette ?
7. Comment expliquer que le résultat de la question 6 n'est pas la somme des valeurs trouvées aux questions 2 et 3 ? Trouver les lignes du texte qui sont responsables de cette différence.
8. Combien de lignes du texte contiennent le symbole + ? Combien de lignes contiennent un point d'interrogation ?
9. Trouver toutes les lignes qui contiennent un chiffre.
10. Trouver toutes les lignes qui contiennent la lettre `z` directement suivie d'une lettre minuscule.
11. Trouver toutes les lignes qui contiennent la lettre `z` directement suivie d'un symbole qui n'est pas une lettre minuscule.
12. Trouver toutes les lignes qui se terminent par une question.
13. Trouver toutes les lignes qui contiennent un mot qui commence par `ab`. (Attention à la casse pour ces mots qui pourraient être en début de phrase.)
14. Combien de lignes contiennent une phrase entre guillemets (incluse entre « et ») qui ne se termine pas sur la même ligne ?
15. Combien de lignes contiennent une phrase entre guillemets qui commence en début de ligne et ne se termine pas sur la même ligne ?
16. Trouver toutes les lignes qui contiennent au moins cinq virgules.
17. Trouver tous les mots (constitués seulement de minuscules et de majuscules) d'une longueur 17 au moins.

2 Sortons des expressions rationnelles

Parfois, les langages rationnels ne suffisent pas pour nos besoins. Par exemple, on ne peut pas trouver les lignes contenant deux fois le même mot séparés par une espace. Pour résoudre ce problème, on utilise des références vers l'arrière, notées `\1`, `\2`, etc.

Si on veut référencer la chaîne de caractères qui a vérifié une certaine sous-expression de l'expression rationnelle, il suffit de mettre cette sous-expression entre parenthèses. Ensuite, `\n` référence la chaîne de caractères qui a vérifié la `n`-ème sous-expression.

Ainsi, dans la commande `grep -E '\<(\w*)\> \<\1\>'`, la référence `\1` aura pour valeur l'expression qui a matché le motif parenthésé `\w*`. Cette commande permet donc de faire la recherche voulue.

Un autre exemple : la commande `grep -E '\<(\w)(\w)(\w)(\w)\>.*\<\1\2\3\4\>'` trouve les lignes qui contiennent deux fois le même mot de longueur quatre.

Avertissement. La reconnaissance des expressions avec références vers l'arrière est difficile et lente (les automates ne suffisent pas) : à utiliser seulement si c'est indispensable !

Exercice 2 *Toujours avec `grep -E`, chercher dans le fichier `swann.txt` toutes les lignes...*

1. ... qui contiennent au moins 15 fois la même voyelle non accentuée ;
2. ... qui contiennent au moins 2 fois un même mot de longueur 10 au moins ;
3. ... où le dernier mot est le même que le premier ;
4. ... qui contiennent deux mots de longueur 3 miroirs l'un de l'autre (par exemple, "abc" et "cba") ;

Question théorique : en tenant compte du fait que l'alphabet est fini, préciser lesquels des points précédents correspondent à des langages rationnels.