Université de Lille — FACULTÉ DES SCIENCES ET TECHNOLOGIES — Département Informatique

Informatique

L3S5 Miage

Analyse Syntaxique – TP1

Expressions régulières d'UNIX

Le système d'exploitation `UNIX` dispose de plusieurs commandes qui utilisent une notation inspirée des expressions régulières pour décrire des motifs. Le shell, en premier lieu, utilise des expressions rationnelles pour construire les arguments des commandes (`ls *.java`). Les expressions régulières sont également utilisées dans différentes commandes :

1. *Le programme de recherche de motif* `grep`[1] *et ses cousins.* La commande UNIX `grep` analyse un fichier et en examine chaque ligne. Si la ligne contient une sous-chaîne équivalente au motif spécifié par une expression régulière, la ligne est alors imprimée sur la sortie standard. Nous utiliserons dans ce TP la commande `egrep` qui autorise la notation complète des expressions régulières, plus quelques extensions supplémentaires.
2. *la commande* `sed` *(Stream Editor)* La commande `sed` permet d'éditer un flux, i.e. de faire une recherche suivie d'un remplacement dans un texte lu sur l'entrée standard, le resultat étant affiché sur la sortie standard (*filtre Unix*). Il existe également la commande `awk` qui est un langage de programmation ciblé sur la reconnaissance de motifs et leur traitement.
3. *Les éditeurs.* La plupart des éditeurs (`vi`[2] ou `emacs` permettent d'analyser un texte pour y trouver une instance d'un motif donné. Le motif est spécifié par une expression régulière (attention, il n'existe pas d'opérateur d'union dans ce cas, mais seulement des classes de caractères). Vous pouvez, par exemple, rencontrer cette fonctionalité dans l'éditeur `emacs` ou dans `ed`.

Les expressions régulières sont également intégrées dans différents langages de programmation par l'intermédiaire de librairies :

1. à l'aide de la classe `RegExp` en JavaScript
2. à l'aide du package `java.util.regex` en Java

La figure 1 présente la syntaxe des expressions régulières en Unix. Voici quelques exemples :

— `^(lundi|mardi|mercredi|jeudi|vendredi)$` correspond à l'une des 5 chaînes lundi...vendredi.
— `^[0-9a-f][0-9a-f]$` correspond à une chaîne de deux caractères choisis parmi les chiffres hexadécimaux
— `^([1-9][0-9]{0,3}|0)$` correspond à une chaîne de 1 à 4 caractères, composée de chiffres uniquement, ne commençant pas par 0, ou bien 0 tout seul.
— `^$` correspond à la chaîne vide.
— `^a*$` correspond à la chaîne vide, ou bien à a ou bien à aa, ou bien à aaa ...
— `^\.$` correspond au caractère . (et non pas à n'importe quel caractère car on a utilisé la déspécialisation `\`).

1. `grep` est l'acronyme de *Globally search for Regular Expression and Print*
2. `vi` est un éditeur en mode console présent dans tout système Unix, et que tout un chacun se doit de connaître.

`c`	un caractère non spécial
`\c`	un caractère spécial non interprété
`^`	le début de la ligne
`$`	la fin de la ligne
`.`	n'importe quel caractère
`[...]`	n'importe quel caractère parmi ...
`[c1-c2]`	n'importe quel caractère compris entre `c1` et `c2`
`[^...]`	n'importe quel caractère ne figurant pas parmi ...
`e1e2`	concaténation des expressions `e1` et `e2`
`e1 | e2`	`e1` ou `e2`
`e*`	zéro ou plusieurs occurrences de `e`
`e+`	une ou plusieurs occurrences de `e`
`e?`	zéro ou une occurrence de `e`
`e{n}`	exactement `n` occurences de `e`
`(e)`	— parenthéser les expressions utilisées avec `|`, `*`, `+`,`?` — associer le texte pour réutilisation (substitution)

FIGURE 1 – Syntaxe des expressions régulières en Unix

1. La commande `egrep`

La commande `egrep` est un filtre[3] qui imprime chaque ligne du fichier d'entrée contenant un *motif* décrit sous la forme d'une expression régulière. La syntaxe de cette commande est :

```
egrep [options] motif fichiers
```

où `options` peut être :

`-n`	impression des numéros de lignes
`-v`	inversion du test (i.e. ne contient pas le motif ...)
`-c`	affichage du nombre de lignes (au lieu des lignes)
`-i`	identification des lettres minuscules et majuscules
`-w`	Le motif s'applique sur des mots entiers : le début du motif s'applique au début de la ligne ou au début d'un mot. De la même manière, la fin du motif s'applique à la fin de la ligne ou à la fin d'un mot.

Exemple

Récupérez le fichier poesies.txt pour essayer ces exemples.

```
egrep -c 'coeur' poesies.txt
```

compte les lignes qui contiennent le mot `coeur`

```
egrep 'coeur' poesies.txt
```

affiche les lignes qui contiennent le mot `coeur`

3. Un filtre est un exécutable qui lit son entrée standard et produit son résultat sur la sortie standard. Ils sont utilisés avec les tubes non nommés (`find -maxdepth 1 -type f -ls | tr -s ' ' | cut -d' ' -f7,11 | sort -n` par ex.)

```
egrep -v '[Ss]' poesies.txt
egrep -vi 's' poesies.txt
```

affiche les lignes qui ne contiennent pas la lettre `s` (majuscule ou minuscule)

Exercice 1. Toujours en utilisant le fichier poesies.txt :

Question 1. *Rechercher les lignes qui commencent par* `A`*.*

Question 2. *Rechercher les lignes qui commencent par* `A` *ou qui terminent par* `t`*.*

Question 3. *Rechercher les lignes qui commencent par une voyelle.*

Question 4. *Rechercher les lignes qui contiennent un* `a`*, un* `e`*, un* `i`*, un* `o`*, un* `u`*, dans cet ordre.*

Question 5. *Rechercher les lignes qui contiennent un* `a`*, un* `e`*, un* `i`*, un* `o`*, un* `u`*, dans cet ordre, et sans autres voyelles intercalées*[4]*. C'est-à-dire qui contiennent comme voyelles que des* `a` *puis que des* `e` *... et ainsi de suite.*

Question 6. *Rechercher les lignes dont un mot au moins contient un* `c` *et un* `i`*, dans cet ordre. Le symbole* `\w` *représente n'importe quel caractère alphanumérique.*

Question 7. *Rechercher les lignes dont un mot au moins comporte exactement 9 lettres.*

2. La commande `sed`

Cette commande est souvent utilisée pour effectuer une recherche suivie d'un remplacement dans un texte lu sur l'entrée standard, le resultat étant affiché sur l'entrée standard. Notez bien que `sed` est un éditeur ligne, i.e. que recherche et remplacement ne concernent qu'une ligne à la fois. La syntaxe d'une substitution est [5]

```
sed -e 's/motifaremplacer/chainederemplacement/'
sed -e 's/motifaremplacer/chainederemplacement/g'
# g pour remplacer toutes les occurences sur une ligne
```

Par exemple la commande

```
echo la nuit tous les chats sont noirs. | sed -e 's/noirs/gris/'
```

affichera à l'écran

```
la nuit tous les chats sont gris.
```

On peut enchaîner plusieurs commandes `sed` en utilisant plusieurs fois l'option `-e`. On peut également placer la suite de commandes `sed` dans un fichier (une commande `sed` par ligne) et utiliser l'option `-f fichier` de `sed` pour exécuter cette suite de commandes.

Lorsque le motif de recherche contient des sous motifs délimités par des parenthèses, on peut utiliser la valeur reconnue correspondante dans la chaîne de remplacement. Par exemple,

4. on ne s'occupe pas des 'y'

5. Pour une description complète des commandes, lisez les pages de manuel avec la commande `man sed` (RTFM).

```
echo lorem ipsum **texte mis en valeur** lorem ipsum |\
sed -e 's/\*\*\(.*\)\*\*/\\033[41m\1\\033[0m/g' | xargs -0 printf
```

donne l'affichage suivant

```
lorem ipsum texte mis en valeur lorem ipsum
```

Dans cet exemple, la chaîne de recherche `\*\*\(.*\)\*\*` correspond à n'importe quel texte (`.*`) situé entre deux étoiles (`\*\*` que l'on doit despécialiser). Le texte reconnu est mémorisé grace aux deux parenthèses qui entourent le motif correspondant (`\(.*\)`). Dans la chaîne de remplacement, ce texte est rappelé en utilisant `\1` (1 est l'ordre d'apparition du sous motif parenthésé dans la chaîne de recherche). Cet exemple utilise les *séquences escape* qui permettent de controler les terminaux : `\033` est le code en octal du caractère escape, et la séquence `\033[` constitue le *Control Sequence Initiator* pour débuter une séquence de controle. Ainsi la séquence d'échappement `\033[41m` change la couleur de fond du terminal en rouge et la séquence `\033[0m` remet celle par défaut (voir ici une description de ces codes). Le `\` doit etre despécialisé dans la commande (d'où `\\033` pour le code escape).

Noter que, lorsqu'il n'y a pas unicité dans la mise en correspondance des motifs, les motifs choisis sont ceux de plus grande longueur. Ainsi,

```
echo lorem ipsum **texte** mis **en valeur** lorem ipsum |\
sed -e 's/\*\*\(.*\)\*\*/\\033[41m\1\\033[0m/g' | xargs -0 printf
```

donne

```
lorem ipsum texte** mis **en valeur lorem ipsum
```

Enfin des classes de caractères sont prédéfinies : `[:punct:]` pour les caractères de ponctuation, `[:blank:]` pour les caractères d'espacement, `[:alpha:]` pour les caractères alphanumériques et `[:digit:]` pour les caractères numériques.

Exercice 2. Récupérez le fichier germinal.txt.zip et décompressez-le avec la commande `unzip`.

Question 8. *Réalisez une commande* `sed` *pour remplacer toute suite de caractères de ponctuation par un espace.*

Question 9. *Réalisez une commande* `sed` *pour remplacer toute suite de caractères d'espacement par un passage à la ligne (*`\n`*).*

Question 10. *Enchaînez ces deux commandes. Qu'obtenez-vous et pourquoi ? Vous remarquez que le résultat contient plusieurs lignes vides qui se suivent, c'est à dire qu'il y a plusieurs caractères* `\n` *consécutifs dans le résultat. Essayez de remplacer toute suite du caractère* `\n` *par un seul* `\n` *en utilisant* `sed`*. Pourquoi n'obtenez-vous pas le bon résultat ?*

Dans ce cas, il faut utiliser la commande `tr -s '\n'`. Pour effectuer un tri de votre résultat et ne conserver qu'une seule occurence, utilisez les commandes `sort | uniq`.

Exercice 3. Récupérez le fichier employees.tsv. Examinez son contenu avec la commande `cat employees.tsv` tout d'abord, puis avec la commande `hexdump -C employees.tsv` ou `od -c employees.tsv`. Vous pouvez constater qu'il s'agit d'un fichier contenant des données au format *tabulation separated values* (le code de la tabulation est `\t`, `09` en hexa). Nous souhaitons créer un fichier au format HTML contenant ces mêmes données pour obtenir cet affichage dans un navigateur.

Question 11. *Réaliser une commande* `sed` *qui crée le corps de la table en* HTML*.*

Note. Les 6 champs de données étant séparés par des tabulations sur une ligne, il vous faut donc reconnaître 6 fois une suite non vide de caractères qui ne sont pas des tabulations, suivie d'une tabulation. `sed` ne dispose pas du `+` pour reconnaître une suite non vide.

Question 12. *Réaliser une commande* `sed` *qui ajoute l'entête* HTML *correspondant à l'exemple. Vous pouvez y placer le style* CSS *utilisé.*

Note. La commande `sed -e 'n i texte'` ajoute une ligne avant la ligne de rang `n` (les rangs commencent en 1). La ligne ajoutée contient le texte `texte`. Si `texte` contient des `\n`, on ajoute en fait plusieurs lignes. `sed -e '/motif/ i texte'` ajoute une ligne avant toutes les lignes correspondantes au motif `motif`.

Question 13. *Réaliser une commande* `sed` *qui ajoute les balises* HTML *fermantes à la fin du résultat.*

Note. La commande `sed -e 'n a texte'` ajoute une ligne après la ligne de rang `n` (les rangs commencent en 1). La ligne ajoutée contient le texte `texte`. `sed -e '/motif/ a texte'` ajoute une ligne après toutes les lignes correspondantes au motif `motif`.

Question 14. *Grouper vos 3 commandes dans un fichier* `tsvtohtml.sed` *et exécuter ce fichier avec la commande* `sed -f tsvtohtml.sed employees.tsv > employees.html` *(on redirige la sortie standard vers le fichier* `employees.html` *qui contient le résultat).*

3. Langages de programmation

Javascript

L'objet `RegExp` est un objet qui représente une expression régulière. Les méthodes `exec(str)` et `test(str)` de ces objets permettent respectivement de renvoyer l'index de la première correspondance trouvée dans l'argument `str`, et un booléen indiquant si une correspondance a été trouvée. Par ailleurs les méthodes `search(regexp)` et `replace(regexp, str)` des objets `String` permettent respectivement de chercher la première correspondance de `regexp` avec l'objet auquel elle est appliquée, et de remplacer les occurences de `regexp` avec l'argument `str` spécifié dans l'objet auquel elle est appliquée. Vous pouvez tester ces différentes méthodes à l'aide de cette page.

La syntaxe des expression régulières en JAVASCRIPT diffère de celle d'Unix. Reportez-vous par exemple à cette page pour les détails.

Exercice 4. Donnez une expression régulière JAVASCRIPT qui vérifie qu'une chaîne correspond à une adresse mail.

html5

HTML5 a introduit l'attribut `pattern` aux éléments `input`. L'attribut `pattern` spécifie une expression régulière avec laquelle la valeur de l'élément `input` doit correspondre. Les expressions régulières sont les mêmes que celles de JAVASCIPT.

Exercice 5. Réalisez une page HTML5 avec un élément `input` n'acceptant que des adresses mail.

Java

Exercice 6. En utilisant les classes `Pattern` et `Matcher` du paquetage `java.util.regex` (Javadoc), écrire un programme qui reconnaît dans les lignes d'un fichier (dont le nom est passé en argument) les URLs HTTP de la forme `http://user:passwd@machine:port/path` et qui affiche, s'ils existent, l'utilisateur, le mot de passe, la machine, le port et le chemin. La syntaxe des expressions rationnelles JAVA est décrite dans la documentation de la classe `Pattern`. Pour lire un fichier ligne par ligne, utiliser la méthode `readLine()` de la classe `BufferedReader`. Vous pourrez utiliser le fichier d'exemples URLExtractor.txt pour tester votre programme.