FAUGERON TYLAN
CHALENCON YOAN
TD 1 TP 3

19,5
20

Utilisation du shell Unix
12 - Expressions régulières

Les expressions régulières (*regular expressions*) ou expressions rationnelles sont utilisées de nombreuses façons dans Unix et plus généralement dans tout système d'exploitation moderne. Elles sont en effet précieuses pour de nombreuses opérations impliquant des chaînes de caractères :

— sélection : comme dans le shell avec par exemple `ls *.[ch]`

— recherche : avec par exemple `grep ".*main(" *.c`

— recherche/remplacement : avec `sed` ou un éditeur de texte : `sed 's-s/:/ /'`

Tous les langages de programmation incluent des procédures et fonctions qui permettent de faire des recherches/remplacements en s'appuyant sur des expressions régulières. Nous allons les manipuler dans trois contextes différents : recherches simples, recherche et remplacement avec un éditeur de texte, puis dans un script Python.

Récupérez et extrayez l'archive `TP-ExpReg.tgz` du répertoire `/users/but/info/Public` (par exemple avec `recup TP-ExpReg`). Cette archive contient quelques fichiers textes que nous utiliserons dans les manipulations qui suivent.

1 Jeux de mots

Le fichier `dico.txt` de l'archive contient une liste de plus de 110000 *formes* du français. On parle de *forme* car un mot du français, comme « aller », peut avoir de nombreuses formes : « va », « vais », « allons », « irez »,...

Vous pouvez ouvrir ce fichier avec votre éditeur de texte favori pour observer son contenu : il contient un mot par ligne et tous les mots sont en majuscules sans accent. Ce fichier a été conçu pour aider dans les jeux de mots : mots croisés, scrabble,....

1. Avec la commande `grep`, faites afficher la liste de tous les mots contenant `MOTTE`

```
grep MOTTE dico.txt
```

2. La liste de tous les mots qui commencent par `'K'`.

```
grep ^K dico.txt
```

3. La liste de tous les mots qui se terminent par `'K'`.

```
grep K$ dico.txt
```

4. La liste de tous les mots qui commencent par `'K'` et se terminent par `'K'`.

```
grep ^K.*K$ dico.txt
```

5. La liste de tous les mots composés d'une seule lettre.

```
grep ^[A-Z]$
```

6. Dans un mot croisé, on cherche un mot de 12 lettres : la première est un `'A'`, la 3e un `'T'`, la 9e un `'V'` et la dernière un `'N'`. Faites afficher tous les mots qui correspondent dans `dico.txt`.

```
grep '^A.T.....V..N$ dico.txt
```

7. Pour jouer au jeu du *5 lettres*, on voudrait la liste de tous les mots de 5 lettres du dictionnaire.

```
grep '^.\{5\}$' dico.txt
```

8. Le problème est qu'on n'a pas droit aux pluriels réguliers. Filtrez avec `grep -v` le résultat de la commande précédente pour éliminer les mots se terminant par `'S'` ou `'X'`.

```
grep '^.\{5\}$' dico.txt | grep -v [SX]$
```

9. On peut obtenir le même résultat sans utiliser `grep -v` en modifiant l'expression régulière de sélection des mots de 5 lettres. Écrivez cette nouvelle version de la commande précédente.

```
grep '^.\{4\}[^SX]$' dico.txt
```

10. Pour terminer, on souhaite connaître pour chaque lettre de l'alphabet combien de mots du dictionnaire commencent par cette lettre. Écrivez un script `compteLettre` qui affiche :

```
A : 10922
B : 6675
C : 15320
...
Y : 38
Z : 179
```

Pour énumérer les lettres de l'alphabet, vous pouvez utiliser :
`for lettre in {A..Z}`
et bien entendu `grep` pour sélectionner les lignes et `wc` pour les compter.

11. On peut obtenir le même résultat (y compris la mise en forme) avec une seule commande combinant dans un tube `cut`, `uniq -c` et `sed`. Donnez cette commande.

```
cut -c 1 dico.txt | uniq -c | sed -E 's/^ *([0-9]*) ([A-Z])$/\2 : \1/'
```

2 Utilisation en programmation

Compilez le programme `txt2dat.c` : vous obtenez des messages d'erreurs. Nous allons les corriger en utilisant le *Rechercher/Remplacer* de l'éditeur Geany. Cette fonctionnalité est accessible par le menu `Rechercher`, entrée `Remplacer...`, (ou plus rapidement avec la touche `Ctrl-H`) : vous ouvrez le dialogue de la figure 1).

NB :

— si vous utilisez d'habitude un autre éditeur que Geany (`gedit`, `gvim`, `vscode`,...), faites les manipulations demandées avec votre éditeur favori, mais indiquez le nom de l'éditeur sur le compte-rendu.

— si votre éditeur favori ne sait pas faire de *Rechercher/Remplacer* avec des expressions régulières, il faut changer d'éditeur favori.

12. Il faut renommer les appels à `strcopy` en appels à `strcpy`.

```
Rechercher : strcopy    Remplacer par : strcpy
```

13. En traduisant son algorithme, le programmeur a oublié qu'en C la transmission du résultat d'une fonction s'écrit `"return res;"` alors qu'en algorithmique on écrit "*nomFonction* ⟸ *res;*". Ouvrez `txt2dat.c` avec Geany et remplacez les occurrences de `"nomFonction <= res;"` par `"return res;"` sans changer la mise en forme. Notez ci-dessous la chaîne

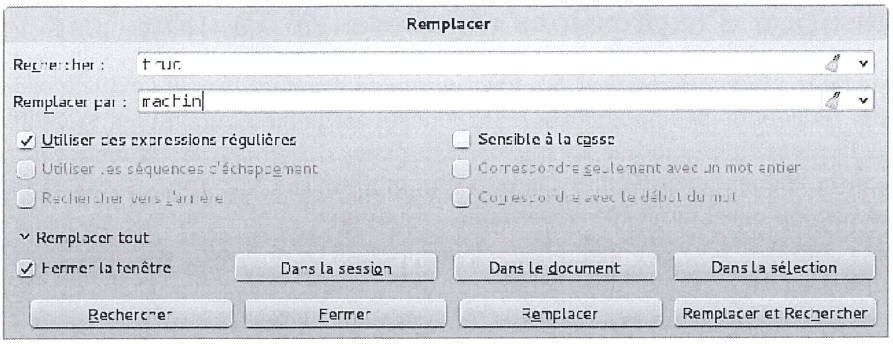

FIGURE 1 – Recherche et remplacement dans Geany

recherchée et la chaîne par laquelle on remplace.

Rechercher: ^(.*).* <= (.*);$
Remplacer par: \1return \2;

14. Enfin, le programmeur a mal lu la documentation et utilisé `open` à la place de `fopen`. De plus il a inversé l'ordre des paramètres.
Il faut donc remplacer `open(X, Y)` par `fopen(Y, X)`.

Rechercher : open\((.*), (.*)\)
Remplacer: fopen(\2, \1)

Finalement, vous devriez pouvoir compiler `txt2dat.c` sans aucune erreur ni avertissement.

Ouvrez le fichier `Page.html` avec Firefox puis avec votre éditeur favori.

16. Donnez l'expression régulière qui permet de chercher une balise HTML : *<texte>*. Pour vérifier, faites une recherche (en général, comme dans Geany, le raccourci est `Ctrl-F`).

Rechercher : <[^/][^>]*>

La description (succincte) des langages de programmation est affichée entre balises `<pre></pre>` mais on voudrait l'afficher sous forme d'une table HTML. La description est structurée avec des `';'` pour séparer les colonnes, on peut donc utiliser une expression régulière pour faire le travail.

17. Geany permet de faire des recherches et remplacement dans la sélection (voir figure 1). Sélectionnez les lignes de description des langages puis faire un rechercher/remplacer de manière à remplacer par exemple :
```
C;Impératif de base;Souple, très souple
```
par
```
<tr><td>C</td><td>Impératif de base</td><td>Souple, très souple</td></tr>
```
Pour que ce soit amusant, il ne faut utiliser qu'une seule expression régulière :

Rechercher : ^(.*);(.*);(.*)$ Remplacer par: <tr><td>\1</td><td>\2</td><td>\3</td></tr>

18. Finalement, remplacer (à la main) `<pre></pre>` par `<table></table>` et vérifiez que vous avez un bel affichage sous forme de table.
19. Pour peaufiner, recherchez/remplacez sur la première ligne du tableau les `td` par des `th`.

3 Utilisation d'expressions régulières en Python

Tous les langages modernes incluent des fonctions pour l'exploitation des expressions régulières. Python offre de nombreuses possibilités mais pour un usage basique, on peut se contenter d'utiliser la fonction `match` du module `re` (*regular expressions*) : `re.match(`*`expression, chaine`*`)` détermine si la chaîne correspond à l'expression, et retourne soit `None` (faux) si la chaîne ne correspond pas, ou un objet si elle correspond. Exemples :

```
print(re.match("<.*>", "une ligne quelconque")) ⇒ None
print(re.match("<.*>", "<html>"))
      ⇒ <re.Match object; span=(0, 6), match='<html>'>
```

L'objet lui-même présente peu d'intérêt mais on peut obtenir la chaîne en correspondance avec :

```
res = re.match("<.*>", "<html>")
print(res.group(0))
```

Si l'expression contient des groupes, on peut obtenir les différents groupes en correspondance avec : `res.group(`*i*`)` ou *i* est le numéro du groupe.

20. Lancez Python puis testez les instructions suivantes en reportant en face de chaque instruction `print` ce qui est affiché :

```
import re          #importation du module re
res = re.match("<.*>", "<h1>Titre</h1>")
print(res.group(0))
res = re.match("(<.*>)(.*)<(.*)>$", "<h1>Titre</h1>")
print(res.group(0))
print(res.group(1))
print(res.group(2))
print(res.group(3))
```

21. Lancez le programme Python `prog.py` fourni. Ce programme parcourt ligne par ligne le fichier `ListeNoms1A.txt` en appliquant une expression régulière à chaque ligne.
22. Créez une copie de ce programme baptisée `courriel.py` puis adaptez la copie de façon à produire la liste des courriels (`Prenom.Nom@etu.univ-grenoble-alpes.fr`), par exemple :
    `Potter:Harry` ⇒ `Harry.Potter@etu.univ-grenoble-alpes.fr`
23. Créez une nouvelle copie : `login.py` qui produise un identifiant constitué des 7 premières lettres (au plus) du nom et de la première lettre du prénom, par exemple :
    `Desproges:Pierre` ⇒ desprogp
    `Ada:Lovelace` ⇒ adal

    Notes :
    - '+' est l'opérateur Python pour la concaténation des chaînes ;
    - si *chaine* est une chaîne Python, *chaine*`.lower()` est la même chaîne en minuscules.

4 S'il vous reste du temps

Dans le script `courriel.py` vous avez peut-être remarqué que certains noms ou prénoms contiennent des espaces. Une adresse de courriel ne pouvant pas contenir d'espace, il faut remplacer les espaces par des tirets ('-'). Il suffit pour ça d'appliquer la méthode `replace` à l'adresse de courriel. Modifiez le script `courriel.py` pour intégrer cette modification.

5 Compte-rendu

Vous rendrez un énoncé complété dans les cases avec les commandes ou expressions régulières utilisées ainsi qu'un listing des scripts, obtenu comme d'habitude avec la commande `imprime` :

```
imprime compteLettre courriel.py login.py
```