Système - TP2
Expressions régulières

1 Expressions régulières simples

Les expressions régulières, aussi appelées REGEX ou REGEXP (pour REGular EXPressions) sont des modèles permettant la recherche de motifs dans des chaînes de caractères. Leur but est souvent d'extraire une information pertinente d'une chaîne de caractère complexe, ou bien d'effectuer un remplacement dans une chaîne.

Presque tous les langages de programmation fournissent des fonctions permettant d'utiliser des regex, et certains programmes tels que grep ou sed sont dédiés à leur utilisation. Les regex sont un outil indispensable à maîtriser pour la manipulation de chaînes de caractères.

Il existe de nombreux types de regex différents. Dans ce TP nous utiliserons les regex ERE (Extended Regular Expressions), qui sont définies par la norme POSIX et très largement reconnues par les différents outils.

Vous trouverez en fin de ce sujet une annexe résumant la syntaxe ERE. Vous n'avez pas besoin d'une machine pour ce premier exercice.

Pour chacune des questions suivantes, donnez les chaînes de caractères matchées par la regex :

Question 1. `[01]{8}`

Question 2. `^(..)*$`

Question 3. `bon{2}`

Question 4. `\l=(\d+|\l)[+-*/%](\d+|\l)`

Question 5. `\u{2}-\d{3}-\u{2}`

Question 6. `([[:alnum:]]+)\.([[:alnum:]]+)@([[:alnum:]]+)\.([[:alpha:]]+)`

Question 7. `\d{1-3}(\.\d{1-3}){3}/\d{2}`

Question 8. `[12]\d{2}(1[012]|0[1-9])(0[1-9]|[1-8]\d|9[0-5])\d{3}\d{3}`

La regex de la question 7 matche certaines chaînes qu'on ne souhaiterait pas matcher (par exemple, `\d{1-3}` matche le nombre 256)... Peut-on faire mieux ?

2 Recherche dans un fichier avec grep

La commande `grep` (Grind Regular ExPression) permet d'utiliser une regex pour filtrer le contenu d'un fichier. `grep` utiliser son propre système de regex, mais peut utilisera les regex ERE si l'on lui donne l'option `-E` : n'oubliez donc pas cette option dans cet exercice.

Exemple d'utilisation de grep avec une regex simple :

```
$ ls
est fichier test test2
$ ls | grep -E "^t"
test test2
```

Pour les questions suivantes, nous allons faire des recherches dans le fichier `/etc/passwd`, qui liste les utilisateurs du système. Il s'agit d'un fichier texte structuré, ce qui facilite la construction de motifs de recherche.

Question 1. Écrivez une commande `grep` qui affiche seulement les utilisateurs dont le shell est `/bin/bash`.

Question 2. Écrivez une commande `grep` qui affiche seulement les utilisateurs dont la description (le 3è champ en partant de la fin) contient la lettre `m`.

Question 3. Écrivez une commande `grep` qui affiche seulement les utilisateurs disposant d'un home différent de `/`.

Question 4. Écrivez une commande `grep` qui affiche seulement les utilisateurs dont l'identifiant est supérieur ou égal à 100.

Pour les questions suivantes, nous allons rechercher des chaînes de caractères dans le fichier `/usr/include/stdio.h`. Il s'agit du fichier d'entête de la libraire standard d'entrée-sortie du C.

Question 5. Écrivez une commande `grep` qui affiche toutes les lignes déclarant une fonction dont le type de retour est `int` (et seulement ces lignes !)

Question 6. Écrivez une commande `grep` qui affiche toutes les lignes déclarant une fonction dont le nom se termine par `printf`.

Question 7. Écrivez une commande `grep` qui affiche toutes les lignes déclarant une fonction dont le premier argument est de type `int`.

Question 8. Écrivez une commande `grep` qui affiche toutes les lignes déclarant une fonction dont le nom du premier argument se termine par `s`.

3 La commande sed

La commande `sed` est un outil puissant qui permet de faire de la manipulation de chaînes de caractères, et notamment du remplacement. Nous utiliserons l'option `-r` qui spécifie à `sed` d'utiliser des ERE. L'option `-n` peut également être utile, elle indique à `sed` de ne pas afficher tout l'input.

`sed` prend en argument une commande parmi les suivantes :
- `p` : print, pour afficher certaines lignes sélectionnées dans l'input (à utiliser avec l'option `-n`) ;
- `d` : delete, pour supprimer certaines lignes de l'input ;
- `s` : substitute, pour remplacer certaines sous-chaînes de l'input.

La commande choisie peut être précédée d'une adresse, qui peut être spécifiée de plusieurs façons :
- `15` : la commande sera appliquée à la ligne 15 du fichier ;
- `15,23` : la commande sera appliquée aux lignes 15 à 23 du fichier ;
- `/regex/` : la commande sera appliquée aux lignes qui matchent le motif regex ;
- `/regex1/,/regex2/` : la commande sera appliquée sur toutes les lignes entre celle qui contient le motif `regex1` et celle qui contient le motif `regex2`.

Si aucune adresse n'est spécifiée, la commande s'applique au fichier entier.

Exemples simples :

```
$ cat fichier
ligne1
ligne2
ligne3
ligne4
ligne5
ligne6
ligne7
$ sed -rn "5p" < fichier
ligne5
$ sed -rn "3,5p" < fichier
ligne3
ligne4
ligne5
$ sed -r "/ligne2/,/ligne6/d" < fichier
```

```
ligne1
ligne7
```

Notez que la commande `d` ne supprime pas les lignes dans le fichier de départ, mais seulement dans l'affichage de `sed`.

Question 1. Écrivez une commande `sed` qui affiche seulement les lignes d'un fichier comportant un nombre de caractères pair.

Question 2. Écrivez une commande `sed` qui affiche seulement les lignes d'un fichier contenant le mot `bonjour`, avec ou sans majuscule.

Question 3. Écrivez une commande `sed` qui supprime les lignes d'un fichier commençant par `//`.

Question 4. Écrivez une commande `sed` qui supprime les lignes vides dans un fichier.

La commande `s` est un peu différente des autres, puisqu'elle est suivie d'un motif de recherche, et d'une chaîne de remplacement, séparés par des `/`. Dans la chaîne de remplacement, il est possible de réutiliser des morceaux de la chaîne matchée par le motif de recherche. Une sous-chaîne peut être extraite à l'aide de parenthèses dans le motif, puis replacée dans la chaîne de remplacement.

Exemples :

```
$ sed -r "s/ligne/texte/" < fichier
texte1
texte2
texte3
texte4
texte5
texte6
texte7
$ sed -r "3,5s/ligne/texte/" < fichier
ligne1
ligne2
texte3
texte4
texte5
ligne6
ligne7
$ sed -r "s/ligne([[:digit:]])/\1ligne/" < fichier
1ligne
2ligne
3ligne
4ligne
5ligne
6ligne
7ligne
```

Notez l'usage des parenthèses dans le dernier exemple : le motif `([[:digit:]])` extrait le chiffre en fin de ligne, et le `\1` dans la chaîne de remplacement replace le chiffre en début de ligne. Chaque paire de parenthèse dans le motif de recherche peut être replacée avec sa référence numérotée de `\1` à `\9`.

Pour les questions suivantes, rendez vous sur le site web de l'UE (`http://perso.ens-lyon.fr/isabelle.guerin-lassous/index-L3RS2P.htm`) puis téléchargez la page web dans un fichier sur votre machine (sous Firefox : `Ctrl+S`). Nous allons faire des recherche et remplacement dans le code HTML du site web.

En utilisant des commandes `sed` et en redirigeant la sortie de la commande vers un nouveau fichier, faites les modifications suivantes dans le HTML :

Question 5. Dans les listes (UL), supprimez chaque élément (LI) qui contient le mot "sécurité".

Question 6. Éditez les coefficients des évaluations de l'UE selon votre préférence – mais avec `sed`, sans éditer le fichier manuellement ! Ne modifiez que le pourcentage, pas le reste de la ligne.

Question 7. Changez la couleur des cellules de tableau (balise TD) dont la couleur est #F0B67F pour la remplacer par une couleur de votre choix.

Question 8. Les dates dans le tableau sont au format jour/mois/année. Passez les au format mois/jour/année, mais seulement pour les séance de Système. Indice : vous pouvez utiliser la couleur.

Question 9. Insérez une nouvelle colonne vide à gauche de chaque colonne existante. Les nouvelles colonnes doivent avoir la même couleur que le reste de la ligne.

4 Les scripts awk

Tout comme `sed`, `awk` est une commande très puissante permettant non seulement d'effectuer des manipulations de chaînes sophistiquées, mais également des calculs complexes sur le contenu d'un fichier.

On utilise souvent `awk` à l'aide d'un script stocké dans un fichier `.awk`. Le script peut alors être exécuté avec `awk -f monscript.awk monfichiertexte`. Le script sera alors exécuté sur le fichier texte fourni.

La syntaxe d'un script `awk` est la suivante :

```
BEGIN { instructions exécutées avant le parcours du fichier }
critère1 { instructions }
critère2 { instructions }
...
critèreN { instructions }
END { instructions exécutées après le parcours du fichier }
```

`awk` parcourt le fichier texte ligne par ligne, et pour chaque ligne, vérifie tous les critères définis dans le script. Si un critère (condition booléenne) est vérifié, alors les instructions associées sont exécutées.

L'un des avantages de `awk` est qu'il découpe automatiquement chaque ligne en champs suivant un délimiteur (pensez à la commande `cut`). Chaque champ est alors accessible via les variables numérotées `$1, $2, $3... $0` correspond à la ligne entière.

Voici un exemple de script `awk` :

```
BEGIN {
    print "Dans le BEGIN, on peut initialiser des variables avant le parcours du fichier, "
    print "mais aussi modifier FS (le séparateur de champs)."
    FS=","
    num=1
}
/^Bonjour/ { print "Cette instruction sera exécutée pour chaque ligne qui commence par Bonjour." }
$2 ~ /^[0-9]{3}$/ { print "Si vous voyez ceci, le second champ est un nombre à trois chiffres." }
! /au revoir\.$/ { print "Si vous voyez ceci, la ligne ne se termine pas par au revoir." }
{ print "Pas de critère ici: cette instruction est exécutée à chaque ligne. " num++ }
$4 > 3 && $4 < 6 { print "Le champ 4 est compris entre 3 et 6." }
{ print "Les champs peuvent être utilisés dans les instructions. Valeur du champ 2: " $2 }
END { print "Fin du parcours, au revoir!" }
```

Les questions suivantes vous demandent d'écrire des scripts `awk` qui seront exécutés sur le fichier `/etc/group`, qui liste tous les groupes système.

Question 1. Écrivez un script `awk` qui affiche le nom (et seulement le nom) de chaque groupe auquel appartient l'utilisateur `root`.

Question 2. Écrivez un script `awk` qui affiche la somme des identifiants (3è champ) de tous les groupes non vides.

Question 3. Écrivez un script `awk` affichant le nom du groupe d'identifiant le plus élevé parmi ceux qui ne contiennent pas l'utilisateur `root`.

Les questions suivantes concernent des scripts `awk` qui analyseront le contenu du fichier `/etc/services`, qui liste des informations sur les ports réseau de la machine. Notez que ce fichier comporte des lignes de commentaires qui doivent être ignorées. Notez également que `awk` accepte une regex comme valeur de FS.

Question 4. Écrivez un script `awk` qui affiche le nombre de services utilisant un port udp, et le nombre de services utilisant un port tcp.

Question 5. Affinez votre script de la question précédente en affichant les même valeurs séparément pour les services dont le port est inférieur ou égal à 1023 (ports connus) et pour les services dont le port est supérieur à 1023 (ports enregistrés).

Question 6. Écrivez un script `awk` qui réécrit le fichier de départ, sans les commentaires, et en listant d'abord tous les services tcp puis tous les services udp.

Question 7. Écrivez un script `awk` qui prend en entrée le fichier produit à la question précédente, et affiche le pourcentage de services qui utilisent à la fois un port udp et un port tcp. Affichez également le pourcentage de services tcp parmi ceux qui n'utilisent qu'un seul port.

On souhaite maintenant utiliser `awk` pour analyser et modifier un fichier `.gpx`. Il s'agit d'un format de fichier contenant une suite de coordonnées GPS, utilisé par exemple pour sauvegarder des parcours de randonnée. Nous utiliserons le fichier `barett_spur.gpx` que vous pouvez trouver ici : `https://www.gpsvisualizer.com/examples/google_gpx.html`. Faites une copie locale du fichier pour pouvoir travailler dessus.

Question 8. Un feu de forêt nous oblige à éviter le carré compris entre les latitudes 45.41 et 45.43 et les longitudes -121.70 et -121.71. Supprimez les étapes concernées du parcours.

Question 9. Finalement, toute la région est inaccessible à cause de l'incendie. On souhaite donc faire notre randonnée ailleurs : décalez tous les points du parcours de 3 degrés de latitude, et 4 degrés de longitude.

Question 10. Écrivez un script `awk` qui affiche les coordonnées GPS du barycentre du parcours décrit dans le fichier (moyenne des coordonnées).

Question 11. On souhaite maintenant produire une nouvelle version du fichier dans laquelle le parcours a été pivoté de 45°dans le sens horaire autour de son barycentre. Pour vous aider : `https://fr.wikipedia.org/wiki/Rotation_vectorielle`. N'oubliez pas de faire un changement de base pour que la rotation soit faite autour du bon centre.

A Annexe : caractères spéciaux dans les expressions régulières

	Description	Exemple d'utilisation
`.`	Motif signifiant "n'importe quel symbole".	`échar.e` matche écharde, écharpe, écharze, échar!e...
`*`	Quantificateur signifiant "n'importe quel nombre de fois, y compris zéro".	`écha*rpe` matche échrpe, écharpe, échaaaaarpe...
`+`	Quantificateur signifiant "au moins une fois".	`écha+rpe` matche écharpe, échaaaarpe, mais pas échrpe.
`?`	Quantificateur signifiant "zéro ou une fois".	`écha?rpe` matche écharpe et échrpe.
`{ }`	Quantificateur personnalisé.	`écha{2}rpe` matche uniquement échaarpe. `écha{1,3}rpe` matche écharpe, échaarpe et échaaarpe.
`^`	Indique le début de la chaîne de caractères.	`^écharpe` matche toute chaîne qui commence par écharpe.
`$`	Indique la fin de la chaîne de caractères.	`écharpe$` matche toute chaîne qui se termine par écharpe.
`|`	OU logique.	`écharpe|écharde` matche écharpe ou écharde.
`\`	Échappement de caractères spéciaux.	`échar\.e` matche échar.e, mais pas écharpe.
`[ ]`	Définit une liste de caractères possibles.	`échar[dp]e` matche écharpe et écharde. `éch[a-c]rpe` matche écharpe, échbrde et échcrde.
`[^]`	Définit une liste de caractères interdits.	`échar[^p]e` matche écharde, écharze, échar"e, mais pas écharpe.
`( )`	Groupe des caractères et permet l'extraction de sous-chaînes (voir l'exercice sur `sed`).	`éch(ar){2}pe` matche échararpe. `éch(arde|elle)` matche écharde et échelle.

B Annexe : classes de caractères

Pour vous simplifier la vie, les regex vous permettent d'utiliser des classes de caractères prédéfinies. Cela vous évite d'avoir trop souvent à écrire `[a-zA-Z0-9]` pour matcher un caractère alphanumérique, par exemple. Certaines classes disposent d'une forme abrégée : vous pouvez utiliser la forme longue ou abrégée indifféremment.

Classe	Raccourci	Caractères matchés
`[:digit:]`	`\d`	Tous les chiffres.
`[:lower:]`	`\l`	Toutes les lettres minuscules.
`[:upper:]`	`\u`	Toutes les lettres majuscules.
`[:alpha:]`		Toutes les lettres, minuscules et majuscules.
`[:alnum:]`		Tous les caractères alphanumériques (lettres et chiffres).
`[:word:]`	`\w`	Tous les caractères alphanumérique, plus l'underscore (_).
`[:blank:]`	`\h`	Espaces et tabulations.
`[:space:]`	`\s`	Tous les caractères d'espacement, y compris le saut de ligne.