Introduction aux systèmes d'exploitation Math-Info
TP n° 8 : Un peu plus de filtres

Retour sur les filtres – « `grep` » et « `cut` »

On revient ici à l'utilisation de tubes et de filtres pour manipuler des données. Pour cela nous aurons besoin de quelque nouvelles commandes, qui sont décrites ainsi que les options nécessaires pour répondre aux exercices à la fin de ce TP. Pour tout complément d'information, « `man` » est votre ami.

Vous trouverez sur moodle un fichier `ratp.csv`. Il provient des données publiques de la RATP (disponibles sur `https://data.ratp.fr/`) et contient des informations sur la fréquentation des diverses stations de son réseau ferré. Il s'agit d'un fichier texte au format CSV (pour *comma-separated values*), représentant un tableau à deux dimensions, les différentes colonnes étant séparées par un séparateur (usuellement une virgule, un point-virgule, une tabulation...)

Exercice 1 – *« `grep` » et la recherche de motifs*

1. Afficher la 1re ligne du fichier pour déterminer les entêtes de colonnes ainsi que le caractère séparateur utilisé. Vous pouvez ensuite afficher par exemple les 10 premières lignes pour voir leur format.

2. Donner les commandes qui permettent d'afficher uniquement les lignes concernant :
   a. les stations qui prennent leur nom d'une mairie ; chercher ensuite quelle option de « `grep` » permet de compter les lignes sélectionnées, puis compter les stations affichées par la commande précédente ; obtenir le même résultat *sans* utiliser l'option précédente.

   b. les stations parisiennes ;

   c. les stations de banlieue ; pour cela, chercher quelle option de « `grep` » permet d'inverser la sélection des lignes ;

   d. les stations qui desservent l'une des portes de Paris ; attention au noms de gares qui contiennent le mot `PORTE` mais ne correspondent pas à des portes de la ville de Paris ;

   e. les stations du 12e arrondissement ;

   f. les stations de la ligne 12 ;

g. les stations de la ville de Sceaux ; attention, il n'y en a que deux, même si trois lignes du fichier contiennent le mot `Sceaux` ; par ailleurs le fichier contient des erreurs de casse, et malheureusement, sans option « `grep` » est sensible à la casse...

h. les stations dont le nom commence par `'M'` ; attention, assurez-vous de ne pas sélectionner aussi les stations dans une ville dont le nom commence par `'M'`. Idéalement, utilisez un seul « `grep` » dans votre réponse.

Exercice 2 – *enchainements de « `grep` » avec d'autres commandes*

Comme réponse, donner une commande qui réalise l'opération demandée.

1. On rappelle que la commande « `ps ax` » permet de lister les processus de tous les utilisateurs, rattachés à un terminal ou non. Essayez à nouveau cette commande et observez le résultat. À l'aide de « `grep` » et d'un tube, afficher seulement les processus de cette liste qui exécutent une commande se trouvant dans `/usr/sbin`.

2. Afficher seulement les processus de la même liste qui ne sont rattachés à aucun terminal.

3. Compter les fichiers de votre *arborescence personnelle* ayant l'extension `.txt`.

4. Lister les références de tous les répertoires contenus dans votre arborescence de racine `~/Cours`.

5. Compter les fichiers ordinaires de votre arborescence personnelle.

6. Éliminer de la sortie de `ls -R` les lignes contenant des `/`.

7. Compter tous les fichiers de votre *arborescence personnelle*...

a. dont la date de dernière mise à jour est aujourd'hui.

b. dont la date de dernière mise à jour est aujourd'hui à partir de 10h00.

c. filtrer juste les répertoires dans la liste précédente.

La commande « `cut` »

La commande « `cut` » est en quelque sorte complémentaire de « `grep` » : si « `grep` » permet de sélectionner/filtrer des lignes, « `cut` » permet de sélectionner/filtrer des colonnes dans des fichiers de type CSV – ou, par extension, pour tout autre fichier texte.
En indiquant à « `cut` » quel caractère joue un rôle de séparateur (*delimiter* en anglais), « `cut` » considére tout ce qui se trouve entre deux de ces séparateurs comme un champ (*field* en anglais). On peut accéder à ces champs par leur numéro (le premier porte le numéro 1), ce qui permet d'en sélectionner un ou plusieurs, comme s'il s'agissait des colonnes d'un tableau.

Exercice 3 – *« `cut` », mais aussi « `sort` » et « `uniq` »*

On reprend le fichier `ratp.csv`.

1. Afficher la liste des villes desservies
   a. telle qu'elle apparaît dans le fichier ;
   b. ✍ triée dans l'ordre alphabétique (avec les doublons éventuels) ;
   c. ✍ triée dans l'ordre alphabétique, sans doublon, en majuscules ;
   d. en faisant précéder chaque ville du nombre de stations qui s'y trouvent ;
   e. ✍ triée par nombre de stations décroissant ;
   f. ✍ triée par nombre de stations décroissant, en séparant les arrondissements de Paris.
2. ✍ Afficher la liste des stations
   a. ✍ triée selon la fréquentation ;
   b. ✍ triée par ordre alphabétique, en affichant *seulement* les 10 stations les plus fréquentées ;
   c. ✍ triée dans l'ordre alphabétique, en n'affichant qu'une seule fois les stations de connexion Métro-RER.

Exercice 4 – *et maintenant tout ensemble*

1. ✍ Afficher *seulement* les stations de connexion Métro-RER.
2. ✍ Afficher *seulement* les stations de connexion entre 3 lignes de métro (au moins).
3. ✍ Afficher *seulement* les stations de connexion entre *exactement* 2 lignes de métro.
4. ✍ Déterminer quel arrondissement possède le plus de stations de la ligne 7.
5. ✍ Afficher, sur une seule ligne, la liste ordonnée des lignes de métro qui desservent le 13^{e} arrondissement.

Travailler sur une arborescence avec « `find` » (optionnel)

La commande « `find` » permet de rechercher des fichiers et éventuellement d'exécuter une action dessus.

> « `find` ***[options] chemins [expression]*** » parcourt les arborescences données par *chemins* en évaluant l'*expression* pour chaque fichier rencontré.

L'expression est constituée de tests et d'actions, tous ces éléments étant séparés par des opérateurs logiques. Quand un opérateur est manquant, l'opération par defaut `-and` est appliquée.

Par exemple, pour trouver les fichiers dont le nom contient `toto` dans l'arborescence de racine le répertoire personnel de l'utilisateur, on utilisera (à l'aide des mêmes jokers que le shell) :

```
find ~/ -name '*toto*'
```

Les principales actions possibles

> « **`-print`** » (action par défaut) liste les fichiers correspondant à la demande.

> « **`-delete`** » supprime chaque fichier correspondant à la demande.

> « **`-exec`** ***cmd*** **`{} ;`** » exécute *cmd* sur *chaque* fichier trouvé.
> « **`-exec`** ***cmd*** **`{} +`** » exécute *cmd* sur *l'ensemble* des fichiers trouvés.

Les fichiers concernés sont désignés par « `{}` ». Par exemple, pour connaître le numéro d'i-nœud de tous les fichiers contenant le motif `toto` de son arborescence, on peut écrire :

```
find ~/ -name '*toto*' -exec ls -i \{\} \;
```

Vous noterez qu'il est nécessaire d'échapper les caractères spéciaux « `{` », « `}` » et « `;` ». Pour cela, on aurait aussi pu écrire :

```
find ~/ -name '*toto*' -exec ls -i '{}' ';'
```

Exercice 5 – *recherche par nom*

1. Lister tous les fichiers de votre arborescence
   - **a.** ✍ d'extension `.java` ;
   - **b.** dont le nom est de longueur 6 ;
   - **c.** dont le nom possède au moins 2 « `.` » ;
   - **d.** ✍ dont le chemin absolu possède au moins 2 « `.` ».
2. ✍ Un usage excessif d'« `emacs` » provoque un trop-plein de fichiers inutiles se terminant par `~`. Écrire une ligne de commande permettant de supprimer tous ces fichiers. ***(Il est fortement recommandé de tester votre ligne de commande sur une sous-arborescence construite pour l'occasion).***

3. ✍ Ôter le droit en exécution pour les utilisateurs autres que le propriétaire sur tous les fichiers d'extension `.sh` de votre arborescence personnelle.

4. ✍ Créer une archive contenant tous les fichiers d'extension `.java`. de votre arborescence personnelle.

Exercice 6 – *autres critères de recherche*

Pour résoudre les questions qui suivent, feuilleter le manuel de « `find` ». Il conviendra de créer le cas échéant des fichiers adhoc pour que la recherche trouve quelque chose...

1. Lister tous vos liens symboliques.

2. Lister tous vos répertoires.

3. ✍ Lister tous vos répertoires vides.

4. ✍ Lister tous vos fichiers de plus de 10 mégaoctets.

5. Lister tous vos répertoires ayant strictement plus que 5 liens.

6. Lister tous vos fichiers ordinaires ayant 2 ou 3 liens, en précisant leur numéro d'inœud.

7. ✍ Lister tous vos liens vers un inœud donné (pour un des numéros obtenus à la question précédente).

8. ✍ Lister tous vos fichiers sur lesquels vous avez tous les droits, tandis que votre groupe et les autres n'en ont aucun.

9. ✍ Lister tous les fichiers qui sont chez vous mais qui ne vous appartiennent pas. L'affichage se fera avec « `ls -l` ».

Les commandes En principe, au cours de cette semaine vous avez utilisé les commandes suivantes :

« `cut [-f` ***champs*****`] [-d`** ***caractere*****`]` »
Les lignes de l'entrée standard sont vues comme des champs (*field* en anglais), délimités par le caractère `TAB` (`'\t'`) par défaut, ou le caractère indiqué par `-d`. La commande affiche sur la sortie les champs indiqués par `-f`.

« `head [-n` ***nblignes*****`]` »
Les *nblignes* premières lignes de l'entrée standard sont affichées. Par défaut *nblignes* vaut 10. Avec l'option `-c` à la place de `-n`, la commande agit sur le nombre de caractères à la place du nombre de lignes.

« `grep [-E] [-i] [-n] [-r] [-v]` ***expression*** **[*références*]** »
permet de selectionner toutes les lignes contenant le motif décrit par *expression*. L'option `-v` inverse la sélection, `-n` permet d'afficher les numéros des lignes sélectionnées, `-i` permet d'ignorer la casse et `-r` permet de chercher récursivement dans tous les fichiers d'un répertoire. Sur certains systèmes l'option `-E` est nécessaire pour utiliser les opérateurs `?` et `+`.

« `sort [-n] [-r]` »
trie les lignes, par ordre alphabétique par défaut, par ordre numérique avec `-n`, et inverse l'ordre avec `-r` (il est également possible de trier sur une autre colonne que la première avec les options `-k` et éventuellement `-t`, mais reportez-vous au `man` pour plus de détails).

« `tail [-n` ***nblignes*****`]` »
Les *nblignes* dernières lignes de l'entrée standard sont affichées. Si le symbole + est placé devant *nblignes*, alors « `tail` » affiche les lignes de l'entrée standard à partie de la ligne numéro *nblignes*. Par défaut *nblignes* vaut 10. Avec l'option `-c` à la place de `-n`, la commande agit sur le nombre de caractères à la place du nombre de lignes.

« `tr [-d] [-s]` ***chaine1*** **[*chaine2*]** »
sans options, l'entrée standard est copiée sur la sortie standard après substitution des caractères spécifiés : un caractère ayant une occurrence dans *chaîne1* est remplacé par le caractère de même position dans *chaîne2*. Des chaînes décrivant des intervalles peuvent également être utilisées, comme `A-Z`, `a-z`, etc.
Avec une seule chaîne, l'option `-d` supprime (*delete*), dans son entrée, les caractères contenus dans la chaîne ; l'option `-s` (*squeeze*) supprime les caractères consécutifs (très utile pour les espaces par exemple : `tr -s ' '`).

« `uniq [-c]` »
recopie l'entrée standard en supprimant les lignes identiques consécutives (elle doit donc généralement être utilisée combinée avec `sort`). Avec l'option `-c`, elle précise le nombre de doublons qu'elle a comptés.

« `wc` »
compte le nombre de lignes, mots et caractères des fichiers ou de l'entrée standard.