TP n°6 – AWK et SED, les filtres programmables

Recopiez les fichiers de `/usr/local/anonymous/SYS/1A/tp6` chez vous. Ils sont également dans `https://perso.univ-rennes1.fr/pierre.nerzic/SYS1A/data/tp6.zip`.

1) AWK

Le fichier `messier` décrit des objets célestes à raison d'une ligne par objet. Chaque ligne contient 5 champs (colonnes) séparés par des virgules :

1. Le premier champ est le numéro de l'objet,
2. le deuxième est un réel qui indique la luminosité de l'objet (on l'appelle la magnitude visuelle, un petit nombre indique un objet brillant, un objet devient invisible à l'oeil nu au delà de 6),
3. le troisième est le nom usuel,
4. le quatrième décrit la nature de l'objet,
5. le dernier champ est la constellation dans laquelle il se trouve.

Exemple : l'objet n°45 a une luminosité de 1.6, on l'appelle les Pléiades, c'est un amas ouvert dans le Taureau.

NB : la liste est déjà triée dans l'ordre des luminosités et il y a un intrus dans cette liste.

Pour faire certaines questions, il faudra peut-être désaffecter la variable LANG en tapant (une fois par séance) : `unset LANG`.

a) Tutoriel

On va commencer par le plus simple : afficher le nombre de lignes du fichier à l'aide de Awk.

i) Le script en paramètre ou dans un fichier

On a le choix : soit taper une ligne de commande comprenant l'appel à Awk et son script :

```
awk 'END { print "il y a " NR "lignes dans le fichier" }' < messier
```

L'autre possibilité, c'est de préparer un fichier contenant les instructions : avec l'éditeur de votre choix, mettez la ligne suivante dans `compter.awk` (attention aux espaces) :

```
END { print "il y a " NR " lignes dans le fichier" }
```

puis lancez l'exécution par :

```
awk -f compter.awk < messier
```

L'avantage du script, c'est de pouvoir l'indenter, le commenter à volonté.

Dans les deux cas, on remarque qu'une chaîne doit être encadrée par des guillemets (pas par des quotes), une variable ne doit pas être encadrée du tout.

Essayez cette variante :

```
END { print "il y a",NR,"lignes dans le fichier" }
```

ii) Séparation des champs

Le langage Awk est destiné à faire des traitements sur des lignes d'informations constituées de différents champs, ils sont automatiquement placés dans les variables spéciales $1, $2… (à ne pas confondre avec celles du Bash). Le caractère séparateur est l'espace par défaut, mais peut être changé en réaffectant la variable FS. Essayer sur messier le script suivant (le taper dans un fichier et lancer l'exécution) :

```
BEGIN { FS = "," }
{ print "l'objet",$1,"est du type",$4 }
```

Modifier ce script pour qu'il affiche le numéro, la luminosité des objets et la constellation d'appartenance.

iii) Opérations numériques

Le langage Awk permet également de faire des calculs sur les données. Voici un script qui calcule différentes informations sur le fichier messier :

```
BEGIN { FS="," ; somme=0 ; nombre=0 }
{ somme += $2 ; nombre++ }
END { print "magnitude moyenne =",somme/nombre" }
```

iv) Conditions sur les actions

Dans les scripts précédents, BEGIN et END sont deux conditions spéciales qui commandent l'initialisation et la terminaison des scripts. On peut aussi spécifier des conditions comme celles du C (sans mettre le if) ou fondées sur des motifs. Essayer le script suivant (aussi étrange qu'il paraisse, n'oubliez pas que ce qui est en début de ligne est une condition comme s'il y avait un if juste avant) :

```
END { print "fin des traitements" }
($1 == 7) { print "objet numéro 7 :",$0 }
BEGIN { FS = "," ; print "début des traitements" }
($2 == 8.4) { print "objet de magnitude 8.4 :",$0 }
/ee$/ { print "objet dont la constellation finit par ee :",$0 }
```

En s'inspirant de celui-ci et en lui enlevant les lignes inutiles, écrire un script qui affiche les nébuleuses. Idem pour afficher les objets d'Ophiuchus. Afficher les objets dont la magnitude est plus petite que 5.5 (si la comparaison des nombres ne marche pas, il faut désaffecter LANG en faisant `unset LANG`).

Afficher les objets qui ont un nom usuel (champ 3 non vide). Afficher uniquement ce nom usuel et la constellation.

Essayer le script suivant qui compte les objets célestes de la constellation du scorpion :

```
BEGIN { FS = "," ; nb = 0 }
($5 == "Scorpion") { nb = nb + 1 ; print "M" $1 " est un " $4 }
END { print "nombre d'objets = " nb }
```

Écrire un script Awk qui compte les objets dont la magnitude est plus petite ou égale à 6 (ils sont visibles à l'œil nu dans un ciel pur).

Écrire un script Awk qui recherche et affiche l'objet messier le plus brillant (magnitude la plus petite). Pour cela, il faut non pas compter les objets, mais mémoriser celui dont le champ n°2 est plus petit que ce qu'on avait trouvé auparavant – et donc initialiser la variable au plus grand nombre possible.

v) Structures de contrôle et tableaux

Awk possède une structure de donnée très puissante : les dictionnaires. On retrouve la même en PHP, Python et Perl. Un dictionnaire c'est une sorte de tableau dont les cases et les indices sont n'importe quoi : des nombres ou des chaînes. Par exemple, je veux créer un tableau appelé SorteDe qui va mémoriser ce qu'est une chose : l'index est une chose, la valeur correspondante est son type. Avec ce tableau, je peux associer poire à fruit, lion à animal comme ceci : SorteDe["poire"]="fruit" ; SorteDe["lion"]="animal". Ensuite, je peux avoir accès à l'association : print SorteDe["lion"] va afficher « animal » et je peux aussi tester la présence d'un élément dans les index : ("poire" in SorteDe) est vrai, ("pomme" in SorteDe) est faux.

Dans le répertoire du TP, il y a deux fichiers, `dictionnaire` et `texte`. Débrouillez-vous pour les concaténer (cat ou éditeur de texte) et envoyer le tout au script suivant. Essayez de comprendre comment il marche. Indication : le mot « fin » qu'on trouve à la fin du dictionnaire déclenche le changement de mode : de la lecture du dictionnaire, ça fait passer au mode traduction.

```
BEGIN { print "Lecture du dictionnaire" ; mode = 1 }
(mode == 1) { DICO[$1] = $2 }
(mode == 2) { if ($1 in DICO){ print DICO[$1]} else { print "???"}}
($1 == "fin") { mode = 2 ; print "Traduction du texte" }
```

Pour comprendre, vous pouvez rajouter cette ligne qui affiche ce qui est lu et la variable mode :

```
{ print "mode=" mode " $0=" $0 }
```

Il faut remarquer que ce programme est un peu extrême : on utilise rarement Awk de cette manière.

Le script suivant est beaucoup plus classique :

```
BEGIN { FS = "," }
($3 != "") { Messier[$1] = $3 }
END { for (num in Messier) { print num " : " Messier[num] } }
```

Maintenant ligne 2, remplacer l'affectation par : `Messier[$3] = $1` et regarder ce que ça affiche...

Essayer ensuite (de comprendre) ce script :

```
BEGIN { FS = "," }
{     if ($1 in Vus) {
            print "M" $1 " est en double"
      } else {
            Vus[$1] = "vu"
      }
}
END { for (n in Vus) { print n, Vus[n] } }
```

Écrire un script Awk qui utilise un tableau pour afficher les types d'objets qui existent (galaxie elliptique, galaxie irrégulière, amas globulaire…) : le tableau doit mémoriser les différents champs n°4 du fichier.

Le modifier pour qu'il compte les types (12 différents) : au lieu de seulement mettre « vu » dans les cases du tableau, il faut incrémenter un compteur.

b) Exercices d'application sur le fichier météoFR

Le fichier météoFR contient des relevés d'observations avec la date, le nom de ville, les températures min et max, un état du ciel, du vent et une hauteur de précipitations. Attention, les questions ne sont pas classées par difficulté.

1. Afficher le nom des villes et l'écart qu'on relève, ville par ville entre leur températures minimale et maximale, puis tout à la fin la température maximale moyenne.
2. Afficher les villes où les précipitations ne sont pas nulles.
3. Afficher par date, le nombre de villes où il y a eu du soleil. Il faut mémoriser un tableau dont les indices sont les dates et les valeurs le nombre de villes où il fait soleil.
4. Afficher les villes qui ont le même état de ciel que la veille (mémoriser les états de ciel de chaque ville dans un tableau et comparer avec l'état précédemment mémorisé).
5. Afficher la température moyenne des villes sur toutes les dates présentes dans le fichier.

2) SED

Sed permet d'éditer un fichier automatiquement. Au lieu de lancer un éditeur interactif comme gedit ou emacs qui montre le fichier en cours de modification, on lance Sed avec des commandes préparées à l'avance. Ces commandes sont exécutées « en aveugle » sur le fichier.

On va travailler sur `config.ini`. C'est un type de fichier très courant, sur Windows et sur Linux, par exemple celui qui configure les partages Windows : `/etc/samba/smb.conf`. Il y a des sections : [nom de la section] et des options : option=valeur, et des commentaires : #... Mais pour ce TP, ça n'a aucune importance.

Voici un exemple où on remplace tous les « oui » par « non » dans le fichier config.ini :

```
sed -re 's/oui/non/' config.ini
```

Le résultat apparaît à l'écran. Si on veut modifier durablement le fichier, alors on rajoute l'option `-i`, mais alors auparavant, faites une copie du fichier config.ini afin de ne pas perdre la version initiale.

Maintenant, remplacez automatiquement tous les « non » par « oui ». Notez que l'une des lignes a été incorrectement modifiée. Trouvez une solution pour que ça n'arrive plus – cette solution est extrêmement importante, il faudra y penser systématiquement (travailler en mode « parano »).

Remplacez enregistrer par enregistrement.

Maintenant, faites en sorte de rajouter des espaces de part et d'autre du signe = dans les lignes nom=valeur.

Maintenant, faites en sorte de changer le nom de la section « affichage » en « display », mais pas dans le commentaire. Ah ah vous avez été piégé ! D'où ça vient ? Alors faites en sorte que ça n'arrive pas.

On peut employer tous les jokers de egrep. Par exemple, on peut remplacer les débuts de section [mot] par -:section:mot:-. La ligne de commande est assez illisible, essayez de comprendre comment elle fonctionne :

```
sed -re 's/^\[([a-z]+)\]$/-:section:\1:-/' config.ini
```

Maintenant, remplacez les lignes « nom=valeur » en mettant à la place « valeur est mise dans nom ».

Une autre application de sed concerne l'ajout de lignes. Essayez cette commande.

```
sed -re '/^arrondi=/a signe=oui\nexposant=non' config.ini
```

La commande sed est structurée ainsi : /motif/a texte… Sauriez-vous rajouter une ligne contenant `dump=non` après celle qui commence par `debug=` ? Sauriez-vous rajouter cette même ligne juste en dessous de `[general]` ? Alors comme ça, vous vous êtes encore fait piéger… il y a des dump=non partout. Heureusement qu'il n'y avait pas l'option -i.

Maintenant une dernière application de sed : supprimer certaines lignes.

```
sed -re '/^arrondi=/d' config.ini
```

Cette commande supprime la ligne en question. Utilisez-la pour supprimer toutes les lignes qui se finissent par `=non`. Pour une fois, il n'y a pas de piège pour ne pas finir sur une mauvaise impression.