Filtres et premiers scripts

Un *filtre* est une commande qui lit les données sur l'entrée standard, effectue des traitements sur les lignes reçues et écrit le résultat sur la sortie standard. Par exemple `cat` est un filtre que vous avez déjà employé. Bien sûr, les entrées/sorties peuvent être redirigées, ou enchainées avec des tubes (pipes).

Pour Unix, un fichier contenant une suite de commandes shell est appelé un *script* et ne nécessite que le droit d'exécution pour pouvoir être exécuté. Pour exécuter un script deux méthodes sont possibles. Soit vous tapez le nom du shell suivi du nom du script à interpréter :

```
$ bash mon-script
```

soit vous tapez directement le nom du script :

```
$ mon-script
```

Dans ce dernier cas, le fichier script doit posséder la permission en exécution et pour garantir son exécution par un shell de type `bash`, la première ligne du fichier indique le chemin absolu du shell qui sera utilisé. La syntaxe de cette ligne est la suivante :

```
#!/bin/bash
```

1 Premiers exemples de script shells

1.1 Exercice Créez avec emacs un fichier exécutable `cmd.sh` contenant :

```
#!/bin/bash
ps -f
```

Dans votre fenêtre `terminal`, lancez la commande `cmd.sh`. Si tout va bien, vous devriez voir s'afficher la liste de vos processus et en particulier un processus (à peu de chose près) de la forme `/bin/bash cmd.sh`.

Un script peut être appelé avec des paramètres :

```
$ mon-script param1 param2 param3
```

Les paramètres sont alors placés dans un vecteur de variables : `param1` dans la première case du vecteur, `param2` dans la deuxième case, `param3` dans la troisième case, etc...
Dans le fichier du script vous pouvez accéder aux éléments du vecteur grâce à des variables : la variable correspondant au premier élément du vecteur est `$1`, celle correspondant au deuxième élément est `$2` et ainsi de suite. La variable donnant la longueur du vecteur est `$#`.

1.2 Exercice Ecrivez le script suivant :

param.sh

```
#!/bin/bash

echo mon premier paramètre est $1.
echo mon deuxième paramètre est $2.
echo mon troisième paramètre est $3.
```

et exécutez la commande :

```
$ param.sh donald riri fifi
```

2 Utilitaires tr, cut, sort

Recopiez chez vous les fichiers `acouper`, `acouper0`, `atrier1`, `blancs_et_tabs`, `caracteres_repetes`, `dates`.

2.1 Exercice Tester le filtre `tr` (**tr**anslate) :

```
$ tr [a-z] [A-Z]
aaaaaaa bbbbbbb ccc 11111 AAA A22222
AAAAAAA BBBBBBB CCC 11111 AAA A22222
$
$ tr [:lower:] [:upper:]
aaaaaaa bbbbbbb ccc 11111 AAA A22222
AAAAAAA BBBBBBB CCC 11111 AAA A22222
```

L'option -d permet de supprimer (delete) des caractères, l'option -s permet de supprimer des répétitions (squeeze-repeats) de caractères, l'option -c [ENSEMBLE] permet d'utiliser les caractères dans le complémentaire de l'ensemble donné. Essayez :

```
tr -s [a] [a] < caracteres_repetes
tr -s [abc] [abc]< caracteres_repetes
tr -d [a]  < caracteres_repetes
tr -d -c [abc]  < caracteres_repetes
```

2.2 Exercice Essayez :

```
cat blancs_et_tabs | wc
expand blancs_et_tabs > blancs_et_tabs1
cat blancs_et_tabs1 | wc
```

Expliquez le résultat.

Faites les questions 1.1, 1.2, 1.3 et 1.4 de la feuille `Problème`.

2.3 Exercice Le filtre `cut` en sélection de colonnes. Tester :

```
cat acouper.
cut -c 4 acouper
cut -c 4,8 acouper
```

Comment faire pour obtenir les colonnes 4 **à** 8 ? les colonnes 4 et suivantes ?

2.4 Exercice Le filtre `cut` en sélection de champs. Tester :

```
cat acouper.
cut -f 1 acouper
cut -f 2 acouper
```

On n'obtient pas les champs escomptés (le premier, puis le deuxième) car le séparateur par défaut est TAB. On essaye alors :

```
cut -d ' ' -f 1 acouper
cut -d ' ' -f 2 acouper
```

Reprenez ces expériences avec le fichier `acouper0`. Comment expliquer ces résultats ? Construisez un fichier `acouper1` sur lequel `cut` agisse "convenablement". Exécutez `cut -d ' ' -f 2 acouper1`

2.5 Exercice Le filtre `sort`. Tester :
```
sort -n -k 3 atrier1
sort -n -r -k 3 atrier1
```
(trie selon le champ 3, et l'ordre des entiers)
```
sort  -k 2 atrier1
```
(trie selon le champ 2, et l'ordre alphabétique).

Remplacez toutes les suites d'espacements (espace ou tabulation) par un caractère ' :'. Le fichier obtenu s'appelle maintenant `atrier2`Ṫester : `sort  -k 3 atrier2`
```
sort -t : -k 3 atrier2
```

2.6 Exercice Ecrire une commande `renverser <fichier>` qui renverse l'ordre des lignes d'un fichier (on pourra combiner `cat -n`, `sort -r` et `cut`)

2.7 Exercice Comment trier, par ordre chronologique, un fichier contenant une date par ligne sous la forme : `jj␣mm␣aaaa` ? Appliquez votre solution au fichier `dates`.

3 Le filtre grep

Une *expression régulière* est une chaîne de caractères, comprenant des caractères ordinaires et des caratères spéciaux. On l'appelle aussi un motif. Ce motif représente de manière compacte un ensemble de chaînes de caractères. On utilise une expression régulière comme paramètre du filtre `grep` mais aussi dans l'utilitaire `sed`. Le tableau ci-dessous contient une partie des caractères spéciaux qui permettent de définir une expression rationelle.

Symbole	**sed**	**grep**	**grep -E**	**signification**
`.`	X	X	X	un caractère quelconque
`*`	X	X	X	expression précédente répétée 0 ou plusieurs fois
`^`	X	X	X	début de ligne
`$`	X	X	X	fin de ligne
`\`	X	X	X	\ placé devant un caractère spécial annule sa signification spéciale
`[ ]`	X	X	X	un caractère parmi ceux de l'ensemble entre crochet
`[^ ]`	X	X	X	tout caractère hormis ceux de l'ensemble entre crochet
`{`n`,`m`}`			X	$n \leq i \leq m$ occurrences de l'expression précédente
$+$			X	expression précédente répétée une ou plus de fois
`?`			X	répétition zéro ou une fois
`|`			X	choix entre deux expressions régulières
`( )`			X	grouper les sous-expressions

EXEMPLES.

motif	**signification**
`mer*`	*me, mer, merr, merrr,...*
`b[aeiuo]g`	deuxième lettre est une voyelle
`^...$`	une ligne contenant exactement trois caractères
`[A-Za-z]`	une lettre
`[^0-9a-zA-Z]`	tout symbole sauf une lettre ou un chiffre
`^[^.]`	le premier caractère n'est pas un point
`^\.[a-z]{2,5}`	une ligne commençant par un point suivi par n lettres minuscules, $2 \leq n \leq 5$, (`grep -E`)
`0{5,}`	cinq ou plus de zéros (`grep -E`)

L'utilitaire `grep` (*General Regular Expression Parser*, analyseur général d'expressions régulières) sélectionne toutes les lignes qui satisfont une expression régulière (ou rationnelle).

`grep` [*options*] *motif fichiers*

Recherche dans un ou plusieurs *fichiers* les lignes qui correspondent à l'expression régulière *motif*. Les valeurs de retour de grep sont : 0 si grep a trouvé des lignes, 1 si grep n'a trouvé aucune ligne, 2 en cas d'erreur. Si les *fichiers* sont absents grep lit l'entrée standard, ce qui permet de l'utiliser comme un filtre.

Quelques options

`-i`	ignorer la distinction entre les majuscules et les minuscules
`-n`	sortir chaque ligne retrouvée précédée par son numéro
`-v`	sortir les lignes qui ne correspondent pas au motif
`-E`	utiliser les expression régulières étendue (comme `egrep`)
`-l`	sortir uniquement les noms de fichiers qui contiennent les lignes recherchées, mais pas les lignes elle-même.
`-e` *motif*	le motif peut être précédé par `-e`, utile si *motif* commence par « - ».
`-w`	sortir les lignes qui possèdent des mots correspondant au motif. Les caractères différents de lettres, chiffres et le caractère « _ » (souligner) sont considérés comme les séparateurs des mots.

3.1 EXERCICE Trouvez parmi vos fichiers dont le nom commence par point ceux qui contiennent le mot `PATH`.

3.2 EXERCICE Récupérez un fichier python contenant des commentaires et des définitions de fonctions (par exemple un fichier que vous utilisez dans le cours de programmation). Quelle commande permet d'extraire toutes les lignes de commentaires ? Ecrivez une commande qui affiche les noms de toutes les fonctions définies dans le fichier. Transformez cette commande en script où le nom du fichier traité sera passé en paramètre.

3.3 EXERCICE Un certain nombre de pages de manuel se trouvent dans `/usr/share/man/man1`. Recherchez dans toutes les pages de manuel des commandes qui commencent par "a" les lignes qui contiennent un des trois mots : "each", "because", "new". Les mots doivent être trouvés même s'ils commencent par une majuscule. Attention, les fichiers sont compressés (voir `zgrep`).

3.4 EXERCICE Récupérer le fichier `exemples_grep.txt`.

- Ecrire une commande qui affiche toutes les lignes du fichier `exemples_grep.txt` qui contiennent le mot `ligne` suivi de deux chiffres.
- Ecrire une commande qui affiche toutes les lignes qui contiennent le symbole `":"` en fin de ligne.
- Ecrire une commande qui affiche toutes les lignes qui contiennent un espace, suivi du symbole `":"`, suivis d'un espace, suivi de cinq caractères quelconques, suivi à nouveau d'un espace.
- Ecrire une commande qui affiche toutes les lignes qui contiennent un espace, suivi d'une virgule, suivie de 0, 1 ou plusieurs caractères quelconques, suivi d'un point.
- Ecrire une commande qui affiche toutes les lignes qui commencent par un caractère différent de l'espace et de `E`.

– Ecrire une commande qui affiche toutes les lignes qui contiennent des mots entiers commençant par `li` et se terminant par un `s`.

4 Application

4.1 Exercice On dispose d'un fichier texte comportant une liste de numéros d'étudiants associés à une note sur 20. Ces deux informations sont séparées par un espace. Les notes sont des nombres entiers à 2 chiffres.

```
$ cat num-note.txt
429221 18
445071 14
444220 20
444454 08
444221 11
444140 14
444129 13
....
```

– Écrire un script `nb_note.sh` qui prend en paramètre une note entre 0 et 20 et qui affiche le nombre d'étudiants qui ont obtenu cette note.

On dispose d'un second fichier comportant une liste de numéros d'étudiants associés à un nom. Ces deux informations sont séparées par un espace.

```
$ cat num-nom.txt
429221 Durand
445071 Dupond
444220 Ellier
444454 Eny
444221 Fall
444140 Fim
444129 Gaud
....
```

– Écrire un script `note.sh` qui prend en paramètre un nom d'étudiant et qui affiche sa note.

5 Utilitaire sed - Pour ceux qui sont en avance

La commande `sed` permet entre autre de substituer une chaine de caractères par une autre dans un fichier. Récupérer le fichier `image.pgm`.

5.1 Exercice Visualisez l'image `image.pgm` avec `gimp` par exemple. Créez un nouveau fichier en remplaçant toutes les occurences de la chaine de caractère “251” par la chaine “0”. Visualisez à nouveau l'image.