Université d'Orléans — L2 Informatique
Outils du développeur — 2018-2019

Contrôle continu

Durée 1h30. Documents autorisés : notes de CM et de TP.

Nom et prénom : **Groupe :**

Exercice 1 Donner le résultat de la ligne de commande suivante :

```
echo "jjjjjr jvyy jjjjjr jvyy ebpppppx lbh" | tr -s 'j' | tr 'a-z' 'n-za-m' | tr -s 'c'
```

`jjjjr jvyy jjjjjr jvyy ebpppppx lbh`	→	Résultat de `echo "jjjjjr jvyy jjjjjr jvyy ebpppppx lbh"`
`jr jvyy jr jvyy ebpppppx lbh`	→	Résultat de `tr -s 'j'` où on ne garde qu'un seul exemplaire de `'j'` consécutifs
`we will we will roccccck you`	→	Résultat de `tr 'a-z' 'n-za-m'` où on transforme les caractères de la chaine `abcdefghijklmnopqrstuvwxyz` en leur correspondants dans la chaine `nopqrstuvwxyzabcdefghijklm` : donc j en w, r en e, etc
`we will we will rock you`	→	Résultat de `tr -s 'c'` où on ne garde qu'un seul exemplaire de `'c'` consécutifs

Exercice 2 La commande `ls -l` affiche le contenu détaillé d'un répertoire, avec des lignes qui ont l'allure suivante :

```
-rw-r--r-- 1 toto staff 284 17 avr 2017 texte.txt
```

où toto est le nom du propriétaire et staff est son groupe d'utilisateurs.

Écrire une ligne de commande qui affiche le nombre de fichiers et de répertoires appartenant à chaque groupe d'utilisateurs.

On va extraire avec `cut` l'information 'groupe du propriétaire' provenant de `ls -l` (donc le 4ème champ). On trie pour avoir les mêmes lignes contiguës, on évoque la commande `uniq -c` pour ne garder qu'un exemplaire de chaque ligne. L'option `-c` fait précéder chaque ligne du nombre de fois où la ligne est apparue dans l'entrée :

```
ls -l | tail -n +2 | tr -s ' ' | cut -d ' ' -f 4 | sort | uniq -c
```

Nous utilisons `tail -n +2` pour ne pas garder la ligne 'total...' qui précède la liste des fichiers.

Exercice 3 Supposons que le fichier `texte.txt` contienne 30 lignes, 300 mots et 5000 caractères. Donner le résultat des commandes suivantes :

a) `wc texte.txt`
Compte le nombre de lignes, de mots et de caractères du fichier.
Affiche donc : `30 300 5000 text.txt`

b) `wc -l texte.txt`
Compte le nombre de lignes du fichier.
Affiche donc : `30 text.txt`

c) `wc -l < texte.txt`
Même chose que précédemment à la différence suivante : L'entrée standard de la commande `wc` est redirigée depuis le contenu du fichier `text.txt`. La commande `wc` voit maintenant un flux provenant de son entrée standard et non plus un contenu provenant du fichier `texte.txt`. D'où l'affichage du résultat (nombre lignes) sans le nom du fichier.

Exercice 4 Écrire une ligne de commande qui affiche le contenu du fichier texte.txt trié par rapport au deuxième mot de chaque ligne dans l'ordre lexicographique inversé. Par exemple, pour

une **ligne** de commande	le résultat affiché sera :	une **ligne** de commande
qui **affiche** le contenu		du **fichier**
du **fichier**		bla **bla**
bla **bla**		qui **affiche** le contenu

`cat text.txt | sort -k2 -r`
La commande `sort` par défaut trie suivant l'ordre lexicographique en prenant toute la ligne comme critère de tri. L'option `-k2` spécifie le champ 2 comme critère de tri. L'option `-r` pour inverser l'ordre.
On pouvait écrire aussi :
`cat text.txt | sort -k2r`
`sort -k2 -r < text.txt`

Exercice 5 Écrire une ligne de commande qui affiche le premier mot qui apparaît le plus souvent dans les lignes 5 à 18 du fichier `texte.txt`. On suppose que le fichier contient un mot par ligne. Utiliser `cat, head, tail, tr, sort` et `uniq`.

`cat test.txt | head -n 18 | tail -n +5 | sort | uniq -c | sort -k1 -nr | head -1`
- `cat test.txt | head -n 18 | tail -n +5` prend les lignes de 5 à 18 du fichier texte.txt
- `sort` pour trier en ordre lexicographique les lignes, ainsi les lignes correspondant à un même mot doivent se suivre
- `uniq -c` pour compter le nombre de lignes dans le groupe pour chaque mot, le résultat est structuré en 2 colonnes, la première le nombre d'occurrences de chaque mot, la deuxième le mot
- `sort -k1 -nr` on trie dans l'ordre décroissant suivant la première colonne, cette fois-ci avec un tri numérique
- `head -1` le premier mot dans la liste triée est celui qui apparaît le plus souvent.

Exercice 6

1. Écrire un script shell `test-fichier` qui précise le type du fichier passé en paramètre et ses permissions d'accès pour l'utilisateur courant si le fichier existe, ou bien indique que le fichier n'existe pas.

   Exemples de résultats :

```
./test-fichier /etc
Le fichier /etc est un répertoire
"/etc" est accessible par root en lecture écriture exécution

./test-fichier /etc/smb.conf
Le fichier /etc/smb.conf est un fichier ordinaire
"/etc/smb.conf" est accessible par jean en lecture
```

Cette version utilise bcp la commande `cut` pour accéder aux différents élément du résultat de `ls -l`. Il existe une autre solution plus simple ne faisant pas appel à l'analyse caractère par caractère du résultat de `ls -l`.

```
#!/bin/bash
type_fichier() {
    # $1 est le nom du fichier/répertoire
    if [ ! -e "$1" ]; then
        # $1 n'existe pas : afficher et quitter avec un code =/= de zéro
        echo "'$1' n'existe pas."
        exit 1
    fi
    # A ce stade, $1 existe. Reste à connaitre ses propriétés
    DROITS=$(ls -ld $1 | tr -s ' ' | cut -d ' ' -f 1)
    PROPRIETAIRE=$(ls -ld $1 | tr -s ' ' | cut -d ' ' -f 3)
    # L'option '-d' de 'ls' se limite au détail du répertoire et non de son contenu
    TYPE=$(echo $DROITS | cut -c 1)
    echo -n "Le fichier '$1' est"
    if [ $TYPE = "-" ]; then
        echo " un fichier ordinaire"
    elif [ $TYPE = "d" ]; then
        echo "un répertoire"
    else
        echo "d'un autre type que 'fichier' ou 'répertoire'"
    fi
```

```
    if [ $(echo $DROITS | cut -c 2) = "r" ]; then LECTURE="lecture"; fi
    if [ $(echo $DROITS | cut -c 3) = "w" ]; then ECRITURE="écriture"; fi
    if [ $(echo $DROITS | cut -c 4) = "x" ]; then EXECUTION="exécution"; fi

    if [ -z "$LECTURE" -a -z "$ECRITURE" -a -z "$EXECUTION" ]; then
        echo "'$1' n'est pas accessible à '$PROPRIETAIRE' que ce soit en lecture, écriture ou exécution"
    else
        echo "'$1' est accessible à '$PROPRIETAIRE' en $LECTURE $ECRITURE $EXECUTION"
    fi
}
type_fichier  "$1"
```

Une autre version :

Cette version utilise la commande 'test' connue plutôt avec son alias '[...]'

```
#!/bin/bash
type_fichier() {
    # $1 est le nom du fichier/répertoire
    if [ ! -e "$1" ]; then
        # $1 n'existe pas : afficher et quitter avec un code =/= de zéro
        echo "'$1' n'existe pas."
        exit 1
    fi
    # A ce stade, $1 existe. Reste à connaitre ses propriétés
    echo -n "Le fichier '$1' est"
    if [ -f "$1" ]; then
        echo " un fichier ordinaire"
    elif [ -d "$1" ]; then
        echo "un répertoire"
    else
        echo "d'un autre type que 'fichier' ou 'répertoire'"
    fi

    PROPRIETAIRE=$(ls -ld $1 | tr -s ' ' | cut -d ' ' -f 3)
    # Il existe une commande 'stat' qui peut extraire plusieurs informations :
    # PROPRIETAIRE=$(stat -f %Su "$1")
    # La chaine format "%Su" permet d'accéder au propriétaire du fichier.

    if [ -r "$1" ]; then LECTURE="lecture"; fi
    if [ -w "$1" ]; then ECRITURE="écriture"; fi
    if [ -x "$1" ]; then EXECUTION="exécution"; fi

    if [ -z "$LECTURE" -a -z "$ECRITURE" -a -z "$EXECUTION" ]; then
        echo "'$1' n'est pas accessible à '$PROPRIETAIRE' que ce soit en lecture, écriture ou exécution"
    else
        echo "'$1' est accessible à '$PROPRIETAIRE' en $LECTURE $ECRITURE $EXECUTION"
    fi
}
type_fichier  "$1"
```

---

2. Écrire un script shell qui affiche le login des utilisateurs dans `/etc/passwd` ayant un numéro d'identification supérieur à 100.
   Une ligne du fichier `/etc/passwd` est constituée de 7 champs séparés par le caractère ' :' :
   (a) nom d'utilisateur, jusqu'à 8 caractères
   (b) le caractère '*'
   (c) un numéro d'identification de l'utilisateur
   (d) un numéro d'identification du groupe
   (e) le nom complet de l'utilisateur
   (f) le répertoire personnel de l'utilisateur
   (g) le compte shell de l'utilisateur.

Exemple : les lignes suivantes sont extraites d'un tel fichier :

```
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
_distnote:*:241:241:DistNote:/var/empty:/usr/bin/false
```

```
lignePasswd() {
  nbLignes=$(cat "/etc/passwd" | wc -l)
  echo $nbLignes
  for i in $(seq $nbLignes); do
    ligne=$(cat "/etc/passwd" | head -n $i | tail -n 1)
    uid=$(echo $ligne | cut -d ':' -f 3)
    user=$(echo $ligne | cut -d ':' -f 1)
    if [ "$uid" -gt 100 ]; then echo $uid $user
    fi
  done
}

lignePasswd
```

Exercice 7 On dispose d'un fichier de données `vols.txt` situé dans notre répertoire personnel. Ce fichier donne les horaires de tous les vols possibles en Europe sur une journée de `Air France,` `KLM` et `Ibéria`. Chaque ligne présente un trajet au format :

`vol:ville_départ:heure_départ:ville_arrivée:heure_arrivée`

où `vol` est le numéro du vol sous la forme `compagnie` avec deux lettres et numéro avec quatre chiffres. Les heures sont au format deux chiffres, le caractère `h` et deux chiffres. Par exemple :
`AF1088:Paris:10h30:Lyon:11h30`
`KL2160:Munich:13h00:Berlin:14h10`
`IB3410:Madrid:07h00:Paris:09h00`
correspondant aux vols
AF1088 (Air France) qui part de Paris à 10h30 et arrive à Lyon à 11h30,
KL2160 (KLM) qui part de Munich à 13h00 et arrive à Berlin à 14h10,
IB3410 (Ibéria) qui part de Madrid à 7h00 et arrive à Paris à 9h00.
Écrire pour chaque question un script shell qui :

1. affiche les numéros de tous les vols qui partent ou arrivent dans une ville dont le nom est donné en paramètre ;

```
afficherVols() {
  # $1 est le nom du fichier
  # $2 est le nom de la ville
  nbLignes=$(cat $1 | wc -l)
  for i in $(seq $nbLignes); do
    ligne=$(cat $1 | head -n $i | tail -n 1)
    depart=$(echo $ligne | cut -d ':' -f 2)
    arrivee=$(echo $ligne | cut -d ':' -f 4)
    if [ "$depart" = "$2" -o "$arrivee" = "$2" ]; then
        numero=$(echo $ligne | cut -d ':' -f 1)
        echo $numero
    fi
  done
}
afficherVols "$1" "$2"
```

Autre solution simple utilisant la commande `grep` :

```
afficherVols() {
  # $1 est le nom du fichier
  # $2 est le nom de la ville
  cat $1 | grep -i $2 | cut -d ':' -f1
}
afficherVols "$1" "$2"
```

On affiche toutes les lignes, on ne garde que celles qui contiennent la ville recherchée (-i pour ignorer la différence maj/min) et on ne conserve que le numéro du vol.

2. affiche toutes les villes triées par ordre alphabétique, qui sont le départ d'un vol Air France ;

```
afficherVilles() {
  nbLignes=$( cat $1 | wc -l)
  for i in $( seq $nbLignes); do
    ligne=$( cat $1 | head -n $i | tail -n 1)
    compagnie=$( echo $ligne | cut -d ':' -f 1)
    chaine=$( echo $compagnie | cut -c 1,2)
    if [ $chaine == "AF" ]; then
       depart=$( echo $ligne | cut -d ':' -f 2)
       echo $depart >> /tmp/resultat
    fi
  done
  cat "/tmp/resultat" | sort
}
afficherVilles vols.txt
```

Autre solution simple utilisant la commande `grep` :

```
cat vols.txt   |
grep -e '^AF'  |  # On ne garde que les lignes qui commencent (donc ^) par 'AF'
cut -d ':' -f2 |
sort
```

3. écrit dans un fichier `nocturnes.txt` sur la première ligne, le nombre de vols dont l'heure d'arrivée est après 19h00 et sur les lignes suivantes, les mêmes renseignements que dans le fichier `vols.txt` concernant ces vols ;

```
nocturnes() {
    while read ligne; do
        heureArrivee=$(echo $ligne | cut -d ':' -f5)
        h=$(echo $heureArrivee | cut -d 'h' -f1)
        if [ "$h" -ge 19 ] || [ "$h" -lt 5 ]; then
            echo $ligne >> /tmp/nocturnes.txt
        fi
    done < vols.txt

    N=$(cat /tmp/nocturnes.txt | wc -l)
    echo $N > nocturnes.txt
    cat /tmp/nocturnes.txt >> nocturnes.txt
    rm /tmp/nocturnes.txt
}
nocturnes
```

Si aucun vol n'arrive dans la tranche recherchée, il n'y aura pas de création de fichier `/tmp/noctures.txt` et la commande `cat /tmp/nocturnes.txt` tombera en erreur. Il serait donc préférable de commencer par le créer vide avec la commande touche `/tmp/nocturnes.txt` On testera également si `N` n'est pas nul avant de créer le fichier nocturnes.txt