Projet Langage de Scripts - SHELL : “Trier en SHELL”

Projet réalisé par Es-sebbani Naim, Guestin Jason, Junhao Li.

Git : https://github.com/Ryujinosorus/Trishell

Description

Réalisation d’un programme en SHELL qui trie les entrées d’un ou plusieurs répertoires.

Installation du programme

Methode 1 (Recommandée):

- sudo ./build.sh

Methode 2:

- chmod u+x src/trishell.sh

Liste des fonctionnalités demandés

- -R : tri le contenu de l’arborescence débutant au répertoire rep. Dans ce cas on triera par rapport aux noms des entrées mais on affichera le chemin d’accès.
- -d : tri dans l’ordre décroissant, par défaut le tri est effectué dans l’ordre croissant.
- -nsdletpg : permet de spécifier le critère de tri utilisé. Ces critères peuvent être combinés, dans ce cas si deux fichiers sont identiques pour le premier critère, le second critère sera utilisé pour les départager et ainsi de suite.
  - -n : tri suivant le nom des entrées ;
  - -s : tri suivant la taille des entrées ;
  - -m : tri suivant la date de dernière modification des entrées ;
  - -l : tri suivant le nombre de lignes des entrées ;
  - -e : tri suivant l’extension des entrées (caractères se trouvant après le dernier point du nom de l’entrée ;
  - -t : tri suivant le type de fichier (ordre : répertoire, fichier, liens, fichier spécial de type bloc, fichier spécial de type caractère, tube nommé, socket) ;
  - -p : tri suivant le nom du propriétaire de l’entrée ;
  - -g : tri suivant le groupe du propriétaire de l’entrée.

Liste des fonctionnalités réalisé

Fonction de tri

La liste des fichiers devant être triés est contenu dans une variable “allData”. Chaque nom de fichier est séparé par un “/”. On prend un nom de fichier puis, à la manière d'un tri par sélection, on le compare aux autres noms de fichiers avec la fonction de tri adéquate pour obtenir une liste parfaitement triée.

Chaque option de tri est représenté dans le programme par une fonction et suit le même principe: La fonction prend deux paramètres (deux noms de fichiers différents) puis: renvoie 1 si $1 < $2 selon les critères de tri renvoie 0 si $1> $2 selon les critères de tri renvoie 2 si $1 = $2 selon les critères de tri

- nameSort() :
  - test $chaine1 < $chaine2 permet de comparer par rapport a l'ordre lexicographique
- sizeSort() :
  - stat -c “%s” renvoie la taille de cette entrée.
- lastChangeSort() :
  - stat -c “%Y” renvoie la derniere modification de cette entrée
- linesSort() :
  - On calcule les lignes seulement sur les entrées de type fichier, soit -f.
  - wc -l – “$fichier” | cut -d -f1 permet de récuperer le nombre de ligne
- extensionSort() :
  - On récupere l'extension grace a `echo -- "$1" | awk -F. '{print $NF}'` puis on trie par rapport a cette dernière
- typeSort() :
  - On regarde le type via -d -f -L -b -c -p -S
- ownerSort() :
  - stat -c “%U” renvoie une chaîne correspondant au nom du groupe de cette entrée.
- groupSort() :
  - stat -c “%G” renvoie une chaîne correspondant au nom du groupe de cette entrée.

Fonctions principales

- getLowest() :
  - Prend trois arguments,`getLowest $file1 $file2 0` compare les deux premiers arguments en fonction du premier paramètre de tri ‘0’, en cas d'égalité il y a un appel récursif `getLowest $1 $2 $(($3+1))`
- getLast() :
  - Prend en paramétre un tableau. Parcourt le tableau et retourne le plus petit grace à `getLowest`.
- change() :
  - `change $tab $elem` renvoie le tableau sans l'elément $elem

- tri() :
  - Prend en paramétre une chaine de caractére représantant un tableau dans lequel chaque élément du tableau est séparé par un /. Parcourt le tableau et affiche le plus petit grace à `getLast` puis réitère cette opération sur le tableau sans le plus petit élément tant que le tableau n'est pas vide.

Récursivité

Pour la récursivité, on place les chemins absolus vers les dossiers séparés par ':' dans une variable `$allFolder` puis on boucle de façon à récuperer les différents chemins en réexecutant le programme grace à $0 en conservant les mêmes paramètres mais avec le chemin vers chaque sous-répertoire.

Liste des fonctionnalités non réalisés

Tri en O(nlogn)

Un tri en 0(nlogn) a été réalisé. Voici le code :

```
getElemAt(){
    echo "$1" | cut -d/ -f$(($2+1))
}


reduce(){
    test $1 = 0 && echo 1 && exit
    res=`bc -l <<< $1/1.3`
    res=`echo $res | cut -d. -f1`
    test -z $res && echo 1 && exit
    if test "$res" = "0"
    then
        echo 1
    else
        echo $res
    fi
}

swap(){
    res=`echo "$1" | sed -E 's/'$2'\//'$3'_\//'`
    res=`echo "$res" | sed -E 's/'$3'\//'$2'\//'`
    res=`echo "$res" | sed -E 's/'$3'_\//'$3'\//'`
    echo "$res"
}
```

```
tri(){
    permutation="true"
    tab=$1
    gap=$2
    while [ $permutation = "true" ] || [ "$gap" -gt 1 ]
    do
        permutation="false"
        gap=`reduce $gap`
        for index in $(seq 0  $(($2-$gap-1)) )
        do
            file1=`getElemAt $tab $index`
            file2=`getElemAt $tab $(($index+$gap))`
            if test `getLowest $file2 $file1 0` = 1
            then
                permutation="true"
                tab=`swap $tab $file1 $file2`
            fi
        done
    done
    IFS=/
}
```

Ceci est une représentation du tri a peigne.
Cependant lors de la permutation d'élements, nous utilisons 3 commandes sed.
A cela s'ajoute l'utilisation de float ainsi que de divisions.
Au final notre tri en O(n2) est plus rapide que notre tri en O(nlogn) qui n'est pas optimisé pour Bash en terme d'exécution.
Nous avons donc conservé pour la version définitive du projet notre tri en O(n2).

Code de retour

Code de retour	valeur
0	Tout s'est bien déroulé
1	Trop d'arguments
2	Doublon dans les paramètres
3	Le chemin spécifié est incorrect
4	Paramètre de tri invalide

Répartition des tâches

	Naim Es-sebbani	Jason Guestin	Junhao Li
Base algorithmique	X		
-n nameSort	X		
-s sizeSort		X	
-m lastChangeSort		X	
-l linesSort		X	
-e extensionSort			X
-t typeSort		X	
-p ownerSort			X
-g groupSort			X
-d Décroissant		X	
-R Récursivité	X		
Testeur			X
Affichage		X	
Débogage	X	X	
Manual	X		

Pourcentage du travail réalisé

Es-sebbani Naim : 35 %
Guestin Jason : 35 %
Junhao Li : 30 %

Conclusion

Ce projet nous a permis de découvrir de nouvelles commandes UNIX tel que: stat, awk ou encore perl.
Nous avons rencontrés de nombreux problèmes (tel que certains caractères spéciaux dans les noms de fichiers qui causaient une erreur dans le programme) mais leurs résolutions ont permis de mieux comprendre le fonctionnement en profondeur des systèmes UNIX.
Nous avons découvert de nouvelles façons de travailler sans l'usage des listes que nous avions généralement l'habitude d'énormément utiliser dans d'autres projets.
Bien que nous aurions aimé mettre en place un algorithme de tri avec une meilleure complexité, nous sommes globalement satisfait du projet car toutes les fonctions implémentées sont opérationnelles.