



## TP shell (4) : synthèse.

### ★ Exercice 1. Petit utilitaire de pense-bête.

L'objectif est d'écrire un utilitaire permettant de stocker et retrouver aisément des bribes d'informations dans un fichier. Par défaut, il stocke les informations passées en argument dans le fichier `~/pensebete`. Si aucun argument n'est passé, il lit les informations au clavier et les stocke dans le fichier avec la commande `cat - > ~/pensebete`. Si le premier argument est `-c`, le script permet de rechercher des informations dans le fichier. L'argument suivant est alors le motif à chercher avec `grep`.

### ★ Exercice 2. Inverser les lignes d'un fichier (réimplémentation de la commande `tac`).

On se propose d'écrire un script `inverse.sh`, qui réécrit un fichier en inversant l'ordre de ses lignes. Ce script reçoit en paramètre le nom du fichier à inverser. Ce paramètre peut être éventuellement suivi du nom d'un autre fichier. Dans ce cas, l'inversion du fichier initial sera effectuée dans le second fichier (le fichier initial n'étant pas modifié). On pourra suivre, étape par étape, les indications suivantes pour parvenir à l'écriture finale du script :

1. Écrire le script `inverse.sh` qui teste si le nombre de ses paramètres est bien égal à 1 ou à 2. Dans le cas contraire, on affichera un message d'erreur. Le code de retour indiquera si l'appel est incorrect.
2. Si le nombre de paramètres est égal à 2, recopier le fichier correspondant au premier paramètre dans le second et relancer le script `inverse.sh`, avec seulement le second paramètre en paramètre (appel récursif). Afin de tester le bon fonctionnement du script, on affichera le message `Un seul paramètre`, dans le cas où le script n'est lancé qu'avec un seul paramètre. Dans le cas d'un appel avec deux paramètres, ce message ne devra apparaître qu'une fois (après l'appel récursif).
3. Dans le cas où il n'y a qu'un seul nom de fichier en paramètre, récupérer dans la variable `nblignes` le nombre de lignes de ce fichier. Tester le script, en affichant la valeur de `nblignes`.
4. Mettre en place la boucle `while` qui permettra de parcourir le fichier ligne par ligne. Pour la tester, on affichera la valeur décrétementée de `nblignes` à chaque parcours de boucle.
5. À chaque parcours de boucle, envoyer les `nblignes` premières lignes du fichier passé en paramètre dans le fichier temporaire `.inv1` (on conseille d'utiliser la commande `head -n $nblignes`, dans laquelle l'argument qui suit l'option `-n` désigne le nombre de lignes à extraire du fichier), puis concaténer la dernière ligne du fichier `.inv1` (extraite avec `tail`) à `.inv2`. Vérifier, en consultant le contenu de `.inv2` après exécution du script. Contrôler qu'il ne manque pas de lignes (ni la première ni la dernière). Effacer `.inv2` après chaque test.
6. Recopier `.inv2` dans le fichier passé en paramètre. Effacer les fichiers `.inv1` et `.inv2`

Tester le script avec un, puis deux fichiers passés en paramètres.

### ★ Exercice 3. Affichage d'informations sur un répertoire.

On souhaite écrire un script donnant diverses indications sur le contenu d'un répertoire. Appelé sans paramètre, ce script doit renvoyer la taille cumulée des fichiers que contient le répertoire courant. Avec l'option `-r`, il doit indiquer le nombre de sous-répertoires. Avec l'option `-f`, il doit renvoyer le nombre de fichiers. Appelé avec un nom de répertoire en paramètre (après les éventuelles options), les valeurs retournées concernent le répertoire indiqué. On pourra suivre les indications suivantes étape par étape :

1. Initialisation :

- Vérifier que le nombre de paramètres est compris entre 0 et 2. Afficher un message d'erreur et produire un code de retour adéquat sinon.
- Initialiser la variable `repertoire` à la valeur `.` (répertoire courant). Initialiser deux variables `dofich` et `dorep` à 0. S'il y a au moins un paramètre et si le premier paramètre commence par un caractère `-` (on utilisera la commande `cut`), afficher le message : `Traitement option`.
- Si le premier paramètre commence par un caractère `-`, identifier les options demandées en affectant la variable `dofich` (respectivement `dorep`) à 1 si l'option `-f` (respectivement `-r`) est active et en la laissant à 0 sinon. Tester sur des exemples en affichant `dofich` et `dorep`.
- Si la présence d'une option autre que `-f` ou `-t` est détectée, afficher un message d'erreur.
- En présence d'une option et s'il y a deux paramètres, affecter la valeur du second à `repertoire`.
- S'il n'y a qu'un seul paramètre (et pas d'option), affecter cette valeur à la variable `repertoire`.
- S'il y a deux paramètres, et s'il n'y a pas d'option, afficher un message d'erreur.

## 2. Traitement :

- Si `dorep` vaut 1, alors :
  - Initialiser `nbr` (variable qui désigne le nombre de répertoires) à 0
  - Parcourir la liste du contenu du répertoire
  - Si un élément `var` de la liste est un répertoire (on pourra effectuer ce test à l'aide de la commande `test -d var`), incrémenter `nbr`
  - Après la boucle, afficher `nbr` et sortir avec un code de retour nul.
- Procéder de même avec les fichiers si `dofich` vaut 1. Tester si `var` est un fichier : `test -f var`.
- Si ni `dorep` ni `dofich` ne valent 1, la commande doit afficher la somme des tailles des *fichiers*. On extraira avec `cut` cette information de la liste détaillée des informations concernant le contenu de repertoire dans une boucle `while`. La somme se calcule avec `total='expr $total + $cefichier'`. Une fois calculée, afficher cette valeur.

### ★ Exercice 4. Et pour terminer... Affichage d'une arborescence de fichiers.

L'objectif est d'écrire un script `arbre.sh` permettant de représenter de la forme suivante la sous-arborescence d'un répertoire passé en paramètre.

Voici une façon de procéder (ce n'est pas la seule):

- Écrire un script affichant tous les sous-répertoires du répertoire en paramètre.
- Le script doit être *récurif* et s'auto-appeler sur chaque sous-répertoire.
- Si le script est invoqué avec deux paramètres, le second est une chaîne de caractères (entre guillemets) qui doit être affichée devant le nom d'un sous-répertoire. Cette chaîne est composée d'espaces, de `|`, de `-` et de `+`
- Lors d'un appel récursif, on ajoute `"| "` au préfixe d'affichage si le répertoire courant a d'autres fils à traiter après cet appel et `" "` s'il s'agit du dernier.
- Lors de l'appel initial, on écrit le nom du répertoire sans préfixe. Cet appel est celui effectué par l'utilisateur (par opposition aux appels récursifs). Le script n'a alors qu'un seul argument.

```
> arbre.sh TEST
TEST
+- FILS1
| +- FILS11
| +- FILS12
| | +- FILS121
| +- FILS13
|   +- FILS131
|     | +- FILS1311
|     +- FILS132
|       +- FILS1321
+- FILS2
| +- FILS21
+- FILS3
| +- FILS31
+- FILS4
  +- FILS41
```

*Attention* : Tester si un répertoire est accessible en lecture et en exécution avant d'y entrer.

Pour vérifier que votre script fonctionne, comparez avec `diff` son affichage à `TP8/exo4/attendu`.