TD1 : Scripts bash

I. Courrier gagnant

1) Écrire un script `couga.sh` qui lit au clavier un nom, prénom, numéro de rue, nom de la rue, code postal et ville, puis affiche sur la sortie standard l'adresse complète de la personne.

2) Le script lit au clavier la somme d'argent gagnée. Si la somme est strictement positive, le script affiche un message du genre "Cher(e) X, vous avez gagné la somme de Y Euros". Si la somme est nulle, le message affiché sera du genre "Cher(e) X, vous n'avez pas gagné cette fois-ci". Pour toute autre valeur saisie, le message affiche un message d'erreur sur la sortie d'erreur et échoue.

3) Le script lit au clavier le nom d'un fichier, puis le crée, sinon il affiche une erreur et échoue.

4) Le script écrit le courrier (adresse et message de gain ou de perte) dans le fichier.

II. Opacification de texte

1) Écrire un script bash `opac.sh` qui demande à lire un texte sur l'entrée standard, jusqu'à la fin de l'entrée standard (`^D`). Le script affiche ensuite le texte en inversant l'ordre des lignes.

2) Le script demande un nom de fichier en entrée et un nom de fichier en sortie. Il vérifie qu'il peut lire le fichier en entrée et créer le fichier en sortie. Enfin il lit le fichier en entrée et le recopie dans le fichier en sortie en inversant l'ordre des lignes.

3) Le script transforme chaque ligne du texte en décalant les lettres : `'a'` ↔ `'n'`, `'b'` ↔ `'o'`, etc (opération appelée "rot13") en utilisant la commande `tr`.

Rappels

La commande `test` admet principalement les options suivantes :

`-d fichier`	vrai si le fichier est un répertoire.
`-e fichier`	vrai si le fichier existe.
`-f fichier`	vrai si le fichier est régulier.
`-s fichier`	vrai si le fichier est non vide.
`-r fichier`	vrai si vous pouvez lire le fichier.
`-w fichier`	vrai si vous pouvez écrire dans le fichier.
`-x fichier`	vrai si vous pouvez exécuter le fichier.
`fichier1 -nt fichier2`	vrai si fichier1 est plus récent que fichier2 (en date de modification).
`fichier1 -ot fichier2`	vrai si fichier1 est plus ancien que fichier2.
`-z chaîne`	vrai si la chaîne est vide.
`-n chaîne`	vrai si la chaîne est non vide.
`chaîne1 = chaîne2`	vrai si les chaînes sont égales.
`chaîne1 != chaîne2`	vrai si les chaînes sont différentes.
`chaîne1 \< chaîne2`	vrai si chaîne1 est inférieure à chaîne2 dans l'ordre lexicographique.
`chaîne1 \> chaîne2`	vrai si chaîne1 est supérieure à chaîne2.
`\( expression \)`	vrai si l'expression est vraie.
`! expression`	vrai si l'expression est fausse.
`expression1 -a expression2`	vrai si les deux expressions sont vraies.
`expression1 -o expression2`	vrai si l'une au moins des expressions est vraie.
`arg1 OP arg2`	Tests arithmétiques, où `OP` est `-eq`, `-ne`, `-lt`, `-le`, `-gt`, ou `-ge`.

TP1 : Scripts bash

I. Une ligne sur deux

Écrire un script `ligpa.sh` qui demande un nom de fichier en entrée. Il vérifie qu'il peut lire le fichier en entrée, sinon affiche une erreur et échoue.

Le script affiche sur la sortie standard toutes les lignes de numéro pair, ceci grâce à une variable que l'on met alternativement aux valeurs `"impair"` ou `"pair"`.

Créer un fichier texte comprenant au moins une dizaine de lignes différentes pour faire des tests (par exemple des lignes commençant par `"un .."`, `"deux .."`, ..). N'oubliez pas de mettre les droits d'exécution au script (`chmod +x ligpa.sh`) avant de le lancer (`./ligpa.sh`) dans le terminal.

II. Début et fin d'un fichier texte

1) Écrire un script `bazar.sh` qui demande un nom de fichier en entrée et un nom de fichier en sortie. Il vérifie qu'il peut lire le fichier en entrée et créer le fichier en sortie.

2) Le script demande un entier `a` et un entier `b`. Il vérifie que $0 \leqslant$ `a` $\leqslant$ `b`, sinon il affiche un message d'erreur et échoue.

3) Le script écrit dans le fichier de sortie les lignes numéro `a` à `b` du fichier d'entrée en se servant des commandes `tail` et `head` reliées par des tubes. Tester sur le fichier d'exemple de l'exercice précédent.

4) Dans votre script, commentez la partie concernant la question précédente ; modifier de façon à ce que le script écrive dans le fichier de sortie toutes les lignes du fichier d'entrée sauf les lignes numéro `a` à `b`, en se servant des commandes `tail` et `head`.

Pour calculer `a-1` ou `b+1` vous pouvez utiliser la commande `expr` redirigée dans un fichier temporaire, dont vous lisez ensuite la première ligne avec `read`.

N'oubliez pas de supprimer les fichiers temporaires avec la commande `rm`.

5) Le script doit maintenant recopier le fichier d'entrée vers le fichier de sortie, de manière à ce que les lignes apparaissent dans l'ordre originel, sauf les lignes numéro `a` à `b` qui apparaîtront triées dans l'ordre lexicographique décroissant (commande `sort`).

Rappels

- ▷ La commande `expr` affiche le résultat d'un calcul passé en argument ; par exemple `expr 4 + 8` affiche `12`. Attention aux espaces.
- ▷ Le shell substitue `$$` par le numéro de processus du shell ; on peut s'en servir pour créer un nom de fichier temporaire unique (par exemple `"tmp-$$.txt"`).
- ▷ `echo -n` affiche une ligne sans retour chariot ; `echo -e` interprète les `"\n"`.
- ▷ `head -`k `[fichier]` affiche les k premières lignes (par défaut $k = 10$) du fichier passé en paramètre (sinon de l'entrée standard).
- ▷ `tail -`k `[fichier]` affiche les k dernières lignes (par défaut $k = 10$) du fichier passé en paramètre (sinon de l'entrée standard). La version `tail -n +`k imprime de la ligne k à la fin.
- ▷ `sort [options]` trie les lignes lues sur l'entrée standard et les recopie sur la sortie standard, selon `option` (par défaut dans l'ordre lexicographique croissant) :
  - `-n` dans l'ordre de la valeur numérique du premier mot ;
  - `-r` dans l'ordre décroissant.