SCR.1.2 TP 07 ⊥ :

Les commandes `expr` et `sort`

I. La commande `expr`.

Elle évalue des expressions de différentes manières, selon la forme de la ligne de commande. Pour chacune des illustrations suivantes, on indique quel en sera le résultat et on vérifie sa réponse en passant la ligne de commande.

Faire `man expr` pour savoir ce que fait une ligne de commande de la forme :

```
expr length STRING
```

Compléter ensuite ce qui suit :

`expr length stock_market`
affiche : .....

Faire `man expr` pour savoir ce que fait une ligne de commande de la forme :

```
expr substr STRING POS LENGTH
```

Ainsi, `expr substr stock_market 1 5`
affiche : .....

et

`expr substr stock_market 7 12`
affiche : .....

Faire `man expr` pour savoir ce que fait une ligne de commande de la forme :

```
expr index STRING CHARS
```

Ainsi, `expr index stock_market "_"`
affiche : ..... parce que .....

et

`expr index stock_market "_c"`
affiche : ..... parce que .....

Une adresse ipv4 (internet protocol version 4) est un mot binaire sur 32 bits. La *notation décimale pointée* d'une adresse ipv4 consiste à écrire chaque octet du mot en décimal, en séparant chaque octet de l'octet suivant par un `"."` (dot).

Exemple : La notation décimale pointée pour
`10110010111000101000011101110010` est `178.226.135.114`

On a utilisé des mécanismes du shell pour faciliter le passage du mot binaire à sa notation décimale pointée. Voici une méthode combinant de tels mécanismes :

1. Dans une variable `addr` du shell, on affecte une suite de `0` et de `1` jusqu'à en avoir 32. On ne compte pas systématiquement pour voir si on est arrivé au bout. On teste au fur et à mesure par `expr` où on en est et on complète en conséquence.
2. On affecte quatre variables : `x`,`y`,`z`,`t` du shell par chacune des sous-chaînes correspondant aux quatre octets dans le mot affecté dans `addr`. On ne fait pas à la main, on fait par `expr` en utilisant le mécanisme de substitution d'une commande par son résultat.

3. On utilise maintenant le fait que le shell évalue des nombres donnés dans n'importe quelle base. Ainsi, `2#101` est compris comme le nombre `101` dans la base `2`. On utilise le mécanisme de l'évaluation arithmétique pour vérifier à quoi le shell évalue un tel nombre :

```
$ echo $((2#101))
5
$ echo $((2#101110))
46
```

4. On affecte alors la variable `addr_dot` du shell par la notation décimale correspondant à `addr`.

II. La commande `sort`.

Elle trie les lignes d'un fichier texte. Elle travaille sur l'entrée standard si le fichier n'est pas indiqué.

1. Un *quidam* nous a envoyé le fichier `ls-output.txt` à trier : (1) par ordre croissant sur la taille des éléments, puis (2) par ordre décroissant sur la taille des éléments.

   Pour chacun des deux cas, on écrit une solution avec une seule ligne de commande.

   Remarque : le fichier `ls-output.txt` est le résultat d'une commande `ls -l`. Or, on remarque que cette option affiche au début une ligne qui renseigne sur un certain total. On peut s'en débarrasser, en combinant la solution avec l'utilisation de la commande `tail`. Ce qui donne une solution avec pipeline. On consultera la page manuel de `tail` pour trouver l'option qu'il faut.

   On refait les questions précédentes de manière à pouvoir envoyer les résultats des tris à l'expéditeur de `ls-output.txt`

2. Le *no-longer-quidam* récidive avec le fichier `find-output.txt` qu'il veut qu'on trie par ordre sur la date de dernière modificaton (1) de la plus récente à la moins récente, puis (2) de la plus ancienne à la moins ancienne.

   Pour chacun des deux cas, on écrit une solution avec une seule ligne de commande.