SCR.1.2 TP 08 ⊥ :

Scripts Shell

La commande `read`

Tout script devra inclure la gestion de l'erreur sur le nombre de paramètres à la ligne de commande.

1. La commande `seq` passée avec un argument numérique affiche la séquence des nombres commençant à 1, et incrémentée de un en un, jusqu'à atteindre la valeur de l'argument. On passe les commandes : `seq 5`, puis `seq 7`.

   (a) A la ligne de commande, on passe une commande composée `for` utilisant la commande `echo`, pour réaliser la même chose que `seq 5`. On recommence pour réaliser la même chose que `seq 7`.

   (b) On veut que l'argument numérique soit passé à un script `my_seq.sh`, qui fait comme `seq` à qui on passe un seul argument. On se restreint au cas où l'argument est un entier positif. On écrit donc `my_seq.sh`.

2. On écrit le script `mult_mat.sh` qui utilise deux arguments numériques entiers positifs `ARG1`, `ARG2`. Il affiche sous forme de matrice la table de multiplication de `{ARG1, ARG1+1,...,ARG2}` par `{ARG1, ARG1+1,...,ARG2}` :

```
$ ./mult_mat.sh 7
Usage: ./mult_mat.sh <NUM_ARG1> <NUM_ARG1>

$ ./mult_mat.sh 7 11
   49    56    63    70    77

   56    64    72    80    88

   63    72    81    90    99

   70    80    90   100   110

   77    88    99   110   121
```

3. On reprend ici un contexte déjà vu dans un TP précédent. Cette fois-ci, en passant par des fichiers. On lit des adresses ipv4, en représentation binaire, à partir d'un fichier `bin_ipv4_addres.dat` qui contient, disons pour simplifier, une adresse par ligne. A l'aide d'un script, on écrit ces adresses en notation décimale pointée et dans un autre fichier.

   (a) `bin2dot-with-for.sh`, prend deux arguments. Le premier est un fichier contenant des adresses ipv4 en représentation binaire. Le deuxième est un fichier qui contiendra les mêmes adresses ipv4 mais en notation décimale pointée. On affectera une variable, disons `addr`, successivement à l'aide d'une commande composée `for`.

   Par exemple :

```
$ cat bin_ipv4_addres.dat
10110010111000101000011101110010
11100101110001010101100101010010
00110010111001101000010001110010
$ ./bin2dot-with-for.sh bin_ipv4_addres.dat
Usage: ./bin2dot-with-for.sh <SRC_FILE> <DEST_FILE>
$ ./bin2dot-with-for.sh bin_ipv4_addres.dat dot_ipv4_addres.dat
$ cat dot_ipv4_addres.dat
178.226.135.114
229.197.89.82
50.230.132.114
$ cat >> bin_ipv4_addres.dat
```

```
11110000111111110000000010101010
$ cat bin_ipv4_addres.dat
10110010111000101000011101110010
11100101110001010101100101010010
00110010111001101000010001110010
11110000111111110000000010101010
$ ./bin2dot-with-for.sh bin_ipv4_addres.dat dot_ipv4_addres.dat
dot_ipv4_addres.dat exists. Overwrite? Yes/No --> Yes
$ cat dot_ipv4_addres.dat
178.226.135.114
229.197.89.82
50.230.132.114
240.255.0.170
```

(b) La commande `read` interagit avec l'utilisateur pour lire une ou plusieurs variables. On fait :

`read x` ; on saisit, disons 7 et on valide (`<CR>`). Tout de suite après, on fait :

`echo "$? x=$x"`. Elle affiche : .....

On recommence avec `read x`, en tapant directement `<CR>`, sans rien saisir. Alors :

`echo "$? x=$x" ;` affiche `-->` : .....

On recommence avec `read x`, sans rien saisir et sans taper `<CR>` mais en tapant `CTRL-D` (rappel : signifie fin du fichier entrée standard). Alors :

`echo "$? x=$x" ;` affiche `-->` : .....

Conclusion sur le code de retour de la commande `read` :

.....

La commande `read` est utile lorsqu'on affecte des variables par lecture dans un fichier. Avec l'illustration qu'on vient de faire, on sait quel est le code d'exit de la commande `read` lorsqu'elle rencontre la fin de fichier.

On écrit un script `bin2dot-with-read.sh`, qui a exactement les mêmes arguments que `bin2dot-with-for.sh`, qui se comporte exactement comme `bin2dot-with-for.sh`, en utilisant les arguments exactement comme le fait `bin2dot-with-for.sh`, mais qui affecte la variable `addr` par une commande composée `while` et la commande `read`.