Programmation et Scripts Shell

.1. *Appel et exécution de scripts shell*

1. Dans votre répertoire personnel, créer le sous-répertoire bin qui sera utilisé pour stocker tous les scripts shell écrits par la suite et se placer dans ce répertoire.

2. Créer le script shell **scriptappel** qui effectue les opérations suivantes :

   - Affecter la chaîne de caractères "abc" à la variable var.
   - Afficher la chaîne de caractères "La variable $var a pour valeur : " suivie du contenu de la variable var.
   - Effectuer une pause de trois secondes.

```
#!/bin/bash
# => le # ! en première ligne du fichier indique le nom de l'interpréteur

var="abc"
echo "La variable var a pour valeur :" $var
sleep 3
```

3. Exécuter le script shell scriptappel de la manière suivante :
   bash scriptappel
   Après la fin de l'exécution du script, quelle est la valeur de la variable var dans votre environnement shell ? Pourquoi ?

```
Cette variable est vide car la variable utilisée dans le script est locale
au script. C'est à dire qu'une fois sorti du script, la variable est détruite
et n'existe plus dans le shell.
Donc quand on affiche la variable « var », sa valeur est nulle car le shell
ne la connaît pas.
```

4. Exécuter le script shell **scriptappel** de la manière suivante :
   scriptappel
   Est-ce possible ? Pourquoi ?

```
Non.
Le fichier script ne peut pas être appelé de cette manière car ce n'est pas
une commande shell connue, le fichier n'a pas l'attribut d'exécution et
n'est pas dans les répertoires spécifiées par PATH.
```

5. Ajouter le droit d'exécution au script scriptappel pour le propriétaire du fichier (Indication : utiliser la commande chmod) et ajouter le chemin de votre répertoire bin à la variable d'environnement PATH s'il n'y est pas déjà.

```
chmod a+x scriptappel
export  PATH=$PATH:$HOME/bin
      HOME est une var d'environnement qui contient le chemin du dossier
personnel
```

6. Exécuter le script shell **scriptappel** de la manière suivante :
   scriptappel
   Après la fin de l'exécution du script, quelle est la valeur de la variable var dans votre environnement shell ? Pourquoi ?

```
Même réponse que pour la question 3.
```

7. Quel shell a été utilisé pour interpréter le script **scriptappel** appelé précédemment ? Faites en sorte que ce soit obligatoirement un shell Bash (Indication : pour imposer un interpréteur plutôt qu'un autre, il suffit de modifier le commentaire spécial en première ligne du script : #! suivi du chemin vers l'interpréteur souhaité).

```
Le shell Bash
```

8. Exécuter le script shell **scriptappel** de la manière suivante :
   . scriptappel
   Après la fin de l'exécution du script, quelle est la valeur de la variable var dans votre environnement shell ? Pourquoi ?

```
Le script s'exécute localement dans le shell. Les variables sont donc créées
dans le shell ce qui permet de les conserver dans l'environnement.
```

9. Exécuter le script shell **scriptappel** de la manière suivante :
   exec scriptappel
   Que se passe-t'il ? Pourquoi ?

```
 Le script s'exécute puis la console se ferme. La commande « exec » ne crée
pas un nouveau processus pour exécuter le script, mais exécute la commande
(le script) en lieu et place du shell en cours. Une fois le script terminé
la console se ferme.
```

.2. Codes de retour

Se reconnecter sur une nouvelle console virtuelle texte en tant qu'utilisateur linux.

1. Taper la commande **ls /etc/passwd** et afficher son code de retour. Expliquer la valeur obtenue.

```
linux@linux-VirtualBox:~$ ls /etc/passwd
/etc/passwd
linux@linux-VirtualBox:~$ echo $?
0
Le « 0 » signifie que la commande s'est déroulée sans problème.
```

2. Taper la commande **ls flop** et afficher son code de retour. Expliquer la valeur obtenue.

```
Le code de retour 2 signifie que la commande retourne une erreur. Plus
précisément que le fichier est non accessible.
```

3. Afficher de nouveau le code de retour de la dernière commande. Est-ce le même ?
   Pourquoi ?

```
Non car le fait d'afficher le résultat d'une commande est une commande en
elle-même. Or la commande précédente s'est bien déroulée ce qui retourne le
code « 0 ».
```

4. Aller dans votre répertoire **bin** et modifier le script **scriptappel** de façon qu'il renvoie un code de retour égal à deux puis le tester (Indication : utiliser **exit**).

```
#!/bin/bash
var="abc"
echo "La variable $var a pour valeur :" $var
sleep 3,
exit 2
```

.3. Enchaînement de commandes

1. Afficher la date système et la liste des fichiers présents dans le répertoire courant en une seule ligne de commande.

```
date ; ls
```

2. Afficher le contenu du fichier **/etc/hosts** s'il existe (Indication : utiliser le code de retour de la commande ls (en éliminant ses sorties standards **stdout** et erreurs **stderr**) pour savoir si **/etc/hosts** existe puis effectuer un enchaînement de commandes conditionnelles avec les caractères **&&**).

```
ls /etc/hosts >/dev/null 2>&1 && more /etc/hosts
Ou plus simple
ls /etc/hosts &&  cat /etc/hosts
```

3. Créer le fichier vide **/tmp/flop** s'il n'existe pas (Indication : utiliser le code de retour de la commande ls (en éliminant ses sorties standards **stdout** et erreurs **stderr**) pour savoir si **/tmp/flop** existe puis effectuer un enchaînement de commandes conditionnelles avec les caractères ||).

```
ls /tmp/flops 2> /dev/null  || touch  /tmp/flops
```

.4. Variables spéciales

1. Dans votre répertoire bin, créer le script shell **scriptvars** qui effectue les opérations suivantes :
   - Afficher le nom du script shell.
   - Afficher le PID du script shell.
   - Afficher le PID du processus père (Indication : utiliser la variable **PPID**).

Chaque affichage doit être précédé d'un énoncé (Exemple pour le nom du script : Mon nom est : "Nom du script shell").

Puis modifier les droits du fichier **scriptvars** de façon à lancer son exécution par la simple saisie de son nom.

```
#!/bin/bash
echo "Mon nom est : $0"
echo "Mon PID est : $$"
echo "Le PPID est : $PPID"
```

```
linux@linux-VirtualBox:~/bin$ PATH=$PATH:$HOME/bin
linux@linux-VirtualBox:~/bin$ scriptvars
```

```
Mon nom est : /home/ubuntu/bin/scriptvars
Mon PID est : 3075
Le PPID est : 2582
```

2. Afficher le PID de votre shell courant puis exécuter le script **scriptvars** des trois manières suivantes :

   bash scriptvars

   scriptvars

   . Scriptvars

```
Shell :
linux@linux-VirtualBox:~/bin$ echo $$
2582

bash  scriptvars :
linux@linux-VirtualBox:~/bin$ bash scriptvars
Mon nom est : scriptvars
Mon PID est : 3098
Le PPID est : 2582

Scriptvars :
linux@linux-VirtualBox:~/bin$ scriptvars
Mon nom est : /home/ubuntu/bin/scriptvars
Mon PID est : 3103
Le PPID est : 2582

.  scriptvars
linux@linux-VirtualBox:~/bin$ . scriptvars
Mon nom est : bash
Mon PID est : 2582
Le PPID est : 2578
```

Les résultats sont-ils ceux attendus ?

```
Oui.
On peut constater que les 2 premiers scripts sont exécutés dans un processus
différent du shell courant. En revanche le « . scriptvars » exécute le script
directement dans le shell ce qui expplique que le PID du script soit celui
du shell.
```

3. Copier le script **scriptvars** en **scriptargs** et modifier ce dernier de façon à :

   - Afficher le nombre d'arguments passés sur la ligne de commandes
   - Afficher les trois premiers arguments de la ligne de commandes

```
#!/bin/bash
echo "Mon nom est " $0;
echo "PID  : " $$;
echo "PPID :  $PPID";
echo "Nombre arguments : $# ";
echo "Arguments 1 2 3 " $1 $2 $3 ;
shift 2
echo "Arguments 1 2 3 " $1 $2 $3
echo "Fin script" $0
sleep 3
```

4. Tester votre script shell **scriptargs** avec les arguments suivants (un seul espace sépare chaque argument) :
   a b c d
   "a b" c d a b c\ d
   a 'b c' d

5. Ajouter les opérations suivantes dans le script scriptargs :
   Décaler les arguments de deux rangs (Indication : utiliser shift).
   Afficher de nouveau les trois premiers arguments Tester de nouveau le script avec les arguments suivants :
   a b c d

.5. Tests de chaînes de caractères

1. Dans votre répertoire **bin**, créer le script shell **scriptchaine** qui effectue les tests suivants sur deux chaînes de caractères passées en argument :

   - Si au moins une des deux chaînes de caractères passées en argument est nulle (deux guillemets ""), sortir avec un code de retour égal à un.
   - Afficher si les chaînes de caractères sont identiques ou non.

```
#!/bin/bash
if [ "$1" = "" ] || [ "$2" = "" ]
    then
        echo "1"
    else
        if [ "$1" = "$2" ]
        then
            echo "Les deux chaines sont identiques";
        else
            echo "$1 est different de $2 ";
        fi
fi
echo "Fin script" $0
sleep 3
```

2. Tester le script shell **scriptchaine** avec les arguments suivants :
   abc ""
   "" abc
   "" ""
   abc abc
   abc ABC
   abc "abc "

.6. Tests et opérations arithmétiques

1. Dans votre répertoire **bin**, créer le script shell **scriptmax** qui retourne le plus grand des deux arguments passés en paramètres.
   Indication : effectuer les opérations suivantes dans le script :

   - Si le nombre d'arguments est différent de deux, sortir avec un code de retour égal à un.

- Afficher le nombre le plus grand ou la valeur du premier en cas d'égalité.

```
#!/bin/bash
if [ $# != 2 ];
    then
        exit 1
    else
        if [ "$2" -lt "$1" ] || [ "$1" = "$2" ]
        then
            echo $1
            exit $1
        else
            echo $2
            exit $2
        fi
fi
echo "Fin script" $0
sleep 3
```

2. Tester le script shell **scriptmax** avec les arguments suivants :

   12

   12 34

   12 6

   12 12

   12 " 34"

```
linux@linux-VirtualBox:~/bin$ scriptmax 12
[9]   Fini                    gedit scriptmax
linux@linux-VirtualBox:~/bin$ scriptmax 12 34
34
linux@linux-VirtualBox:~/bin$ scriptmax 12 6
12
linux@linux-VirtualBox:~/bin$ scriptmax 12 12
12
linux@linux-VirtualBox:~/bin$ scriptmax 12 " 34"
34
```

3. Dans votre répertoire **bin**, créer le script shell **scriptmin** qui retourne le plus petit des deux arguments passés en paramètres.
   Indication : effectuer les opérations suivantes dans le script :

   - Si le nombre d'arguments est différent de deux, sortir avec un code de retour égal à un.
   - Afficher le nombre le plus petit ou la valeur du premier en cas d'égalité.

```
#!/bin/bash
if [ $# != 2 ];
    then
        exit 1
    else
        if [ "$2" -gt "$1" ] || [ "$1" = "$2" ]
        then
            echo $1;
            exit $1;
        else
            echo $2;
```

```
            exit $2;
        fi
fi
echo "Fin script" $0
sleep 3
```

4. Tester le script shell scriptmin avec les arguments suivants :

   12

   12 34

   12 6

   12 12

   12 " 34"

```
linux@linux-VirtualBox:~/bin$ scriptmin 12
linux@linux-VirtualBox:~/bin$ scriptmin 12 34
12
linux@linux-VirtualBox:~/bin$ scriptmin 12 6
6
linux@linux-VirtualBox:~/bin$ scriptmin 12 12
12
linux@linux-VirtualBox:~/bin$ scriptmin 12 " 34"
12
```

.7. Boucle while

1. Écrire le script shell **scriptcompar** qui affiche le plus petit et le plus grand nombre parmi les arguments passés sur la ligne de commandes. Le nombre d'argument doit être supérieur à un (Indication : Réutiliser les scripts **scriptmax** et scriptmin précédemment écrits. Utiliser une boucle **while** avec la commande **shift** pour traiter l'ensemble des arguments).

2. Tester le script shell **scriptcompar** avec les arguments suivants :

   1

   1 2

   1 2 3

   3 5 2 6 1 8 7

```
#!/bin/bash
if [ $# -lt 1 ];
    then
        exit 1;
    else
        nb="$#"
        min="$1"
        max="$2"
        i=1
        while [ $i -lt "$nb" ]
        do
# ce script est base sur les resultats de stdout
            min=$(scriptmin $min $2)
            max=$(scriptmax $max $2)
            echo "Minimun $min Maximun $max"
            shift
            let i=$i+1
        done
```

```
fi
echo "Fin script" $0
sleep 3
```

```
#!/bin/bash
if [ $# -lt 1 ];
    then
        exit 1;
    else
        nb="$#"
        min="$1"
        max="$1"
        i=1
        while [ $i -lt "$nb" ]
        do
# ce script est base sur les codes de retour des commandes
            scriptmin $min $2
            min=$?
            scriptmax $max $2
            max=$?
            echo "Minimun $min Maximun $max"
            shift
            let i=$i+1
        done
fi
echo "Fin script" $0
sleep 3
```