Scripting Shell GNU/Linux – Compléments

Compte Rendu : votre CR contiendra les réponses aux questions posées à déposer avant la fin de la séance dans le répertoire à votre nom du dépôt B2B3_AdminSystemes-PC/Gnu-Linux.

Nom du fichier : **05-ScriptingShellMenus**

N'hésitez pas à tester les nouvelles commandes (et celles que vous ne connaissez pas bien) directement sur la ligne de commande pour comprendre son fonctionnement avant de l'inclure dans un script.

1. Sortie de script

La commande **exit** permet de mettre fin à un script **à tout moment**. Par défaut la valeur retournée est **0 (pas d'erreur)** mais n'importe quelle autre valeur de 0 à 255 peut être précisée. Vous récupérez la valeur de sortie par la variable **$?**. L'intérêt de pouvoir retourner la valeur que l'on veut est qu'on peut faire un **exit nnn** lorsqu'on a « trappé » une erreur dans le script à l'aide d'une structure alternative.

- Rajoutez l'instruction « exit 122 » à un script précédent. Exécutez-le et examinez le contenu de « $? »

```
Echo $?
```

2. Tests sur les fichiers

Il est possible de tester l'existence et les propriétés des fichiers. La syntaxe est : **test option nom_fichier**

- Et les options sont les suivantes. Complétez les rôles nos précisés avec l'aide en ligne ($ man test).

Option	Rôle	Option	Rôle
-f	Fichier normal.	-x	
-d	Un répertoire.	-s	
-c	Fichier en mode caractère.	-e	
-b	Fichier en mode bloc.	-L	Le fichier est un lien symbolique.
-r		-u	Le fichier existe, SUID-Bit positionné.
-w		-g	Le fichier existe SGID-Bit positionné.

2.1. Tests combinés par des critères ET, OU, NON

- Vous pouvez effectuer plusieurs tests avec une seule instruction.

Critère	Action
-a	AND = ET logique
-o	OR = OU logique
!	NOT = NON logique

- Testez et expliquez la commande suivante :

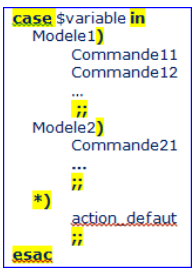
test -f "e.sh" -a -x "e.sh" && echo "OK" || echo "KO"	

2.2. Syntaxe allégée avec les crochets

- Testez et expliquez la commande suivante :

[-f "e.sh" -a -x "e.sh"] && echo "OK" || echo "KO"	

3. Choix multiples case



La commande **case ... esac** permet de vérifier le contenu d'une variable de manière multiple (équivalent à plusieurs « if » imbriqués mais plus simple à écrire) :

Surligné en jaune, la syntaxe de la commande.

Le **modèle** est soit un simple texte, soit composé de caractères spéciaux, terminé par une parenthèse fermante. Dans l'exemple de la page suivante, les caractères spéciaux sont « * » pour « a* » (commence par la lettre a) et « [123] » dans « fic[123] » (qui signifie soit fic1, soit fic2, soit fic3).

Chaque bloc de commandes lié au modèle doit se **terminer par deux points-virgules**. Dès que le modèle est vérifié (vrai), le bloc de commandes correspondant est exécuté. **L'étoile** en dernière position est **l'action par défaut** si aucun critère n'est vérifié. Elle est facultative.

- Saisissez, testez et commentez le script suivant. Attention, les caractères « * » ont des significations différentes.

case1.sh	**Exécutez-le et notez et expliquez les résultats**
#!/bin/bash case $1 in a*) echo "Commence par a" ;; b*) echo "Commence par b" ;; fic[123]) echo "fic1 fic2 ou fic3" ;; *) echo "Commence par un autre caractère" ;; esac exit 0	./case1.sh "au revoir" ./case1.sh bonjour ./case1.sh fic2 ./case1.sh erreur

- Ecrivez un script qui traduit en anglais un chiffre passé en paramètre. Les chiffres sont compris entre "zero" et "cinq". Le script répondra "inconnu" en cas d'erreur.

TradFrAnglais.sh

4. Saisie de l'utilisateur

La commande **read** permet à l'utilisateur de saisir une chaîne de caractères et de la placer dans une ou plusieurs variables. La saisie est validée par [Entrée].

read var1 [var2 ...]

Si plusieurs variables sont précisées, le premier mot ira dans var1, le second dans var2, et ainsi de suite. S'il y a moins de variables que de mots, tous les derniers mots vont dans la dernière variable.

- Saisissez, testez et commentez le script suivant.

read.sh	**Exécutez, notez et commentez les résultats**
#!/bin/bash echo –e "Continuer (O/N) ? \c" read reponse echo "reponse=$reponse" case $reponse in O) echo "Oui, on continue" ;; N) echo "Non, on s'arrête" exit 0 ;; *) echo "Erreur de saisie (ni O, ni N saisi)" exit 1 ;; esac # echo –e " Tapez deux mots ou plus :\c" read mot1 mot2 # echo –e "mot1=$mot1\nmot2=$mot2" exit 0	Utilité des lignes 2, 3 et 4 ? Pour bien comprendre, vous pouvez les exécutez en ligne de commande. ./read.sh Continuer (O/N) ? O ./read.sh Continuer (O/N) ? N ./read.sh Continuer (O/N) ? x Quelle est l'utilité de "-e" et "\c" ? → $man echo

- Ecrire un script qui demande deux nombres à l'utilisateur et qui affiche la somme et la moyenne. Vous aurez besoin de typer vos variables en entier par la commande typeset.
  - Dans un premier temps, vous n'utilisez pas ici la structure case…esac, seulement la commande read.
  - Ensuite vous modifierez votre script pour demander à l'utilisateur s'il veut continuer ou arrêter.

5. Les boucles

Elles permettent la répétition d'un bloc de commandes soit un nombre limité de fois, soit conditionnellement. Toutes les commandes à exécuter dans une boucle se placent entre les commandes **do** et **done**.

5.1. Boucle for

```
for var in liste
do
   commandes à exécuter
done
```

La boucle **for** ne se base pas sur une quelconque incrémentation de valeur mais sur une liste de valeurs, de fichiers...

La liste représente un certain nombre d'éléments qui seront successivement attribués à var.

- Saisissez, testez et commentez le script **for1.sh** suivant.

Avec une variable : for1.sh	**Exécutez-le et notez les résultats et commentaires**
#!/bin/bash for params in $@ do echo "$params" done	./for1.sh test1 test2 test3
Modifier ce script pour qu'un numéro séquentiel s'affiche devant chaque argument. On doit avoir **1:test1** **2:test2…**	

Si vous ne précisez aucune liste à for, alors c'est la liste des paramètres qui est implicite. Ainsi le script précédent aurait pu ressembler à :

- Saisissez, testez et commentez le script suivant.

Liste implicite : for1-2.sh	**Exécutez-le et notez les résultats et commentaires**
#!/bin/bash for params do echo "$params" done	./for1-2.sh test1 test2 test3

- Méthodes pour compter de 1 à n avec une boucle for. Commentez les trois commandes suivantes :

Commandes	**Commentaires**
seq 5	
for i in $(seq 5); do echo $i; done	
for ((a=1 ; a<=5 ; a++)); do echo $a; done	Boucle "traditionnelle" du langage C

5.2. break et continue

La commande **break** permet **d'interrompre une boucle**. Dans ce cas le script continue après la commande **done**. A ne pas confondre avec la commande **exit** qui **interrompt le script**.

- Saisissez, testez et commentez le script. Renseignez-vous auparavant sur le paramètre –z de la commande test.

une liste d'éléments explicite : while1.sh	**Testez et répondez aux questions suivantes**

while true
do
  echo -e "Chaine ? \c"
  read a
  if [-z "$a"]
  then
  break
  fi
done
echo « Suite du script »
	Que fait ce script ?

6. Boucle select : création de menu

```
select variable in liste_contenu
do
   traitement
done
```

La commande **select** permet de créer des menus simples, avec sélection par numéro.

La saisie s'effectue au clavier avec le prompt contenu dans la variable **PS3**.
Si la valeur saisie est incorrecte, une boucle s'effectue et le menu s'affiche à nouveau. Pour sortir d'un select il faut utiliser un **break**.

➢ Saisissez, testez et commentez le script suivant.

une liste d'éléments explicite : select.sh	**Testez et répondez aux questions suivantes**

#!/bin/bash
PS3="Votre choix :"
echo "Quelle donnee ?"
select reponse in Jules Romain Francois quitte
do
  if [["$reponse" = "quitte"]]
  then
  echo "Vous voulez arrêter"
  break
  fi
  echo "Vous avez choisi $reponse"
done
echo "Au revoir."
exit 0
	Dans quel cas le script affiche « Au revoir » ?
	
	Modifiez le script en utilisant une variable choix que vous initialiser avec les 4 choix (choix="Jules Romain…") et que vous utilisez après le « in » du select.

➢ Ecrivez un script qui affiche un menu donnant le choix entre 3 commandes :
- Affichage de la date en français
- Addition de deux nombres
- Quitter

Menu.sh

7. Les fonctions

Les fonctions sont des bouts de scripts nommés, **directement appelés par leur nom**, pouvant accepter des paramètres et retourner des valeurs. Les noms de fonctions suivent les mêmes règles que les variables.

funtion nom_fonction ()
{
  commandes
  return
}
	Parcourez ce tutoriel :
	http://www.tuteurs.ens.fr/unix/shell/fonction.html
	
	Et testez les scripts proposés

Les fonctions peuvent être soit tapées dans votre script courant, soit dans un autre fichier pouvant être inclus dans l'environnement. Pour cela saisissez :

➢ Saisissez le fichier « exFonct » suivant, **sans extension et sans changer les droits**.

exFonct	**Testez et répondez aux questions suivantes**
function l1 () {	Tapez les 5 commandes suivantes dans l'ordre : l1 # C'est la lettre L minuscule comme ls…

ls -l $@
}
 	
function l2 ()
{
ls -i $@
}
	l2
	. exFonct # Attention « espace » après le point !
	l1
	l2
	Explications ?

Complément sur les fonctions :

- Un fichier de fonctions ne **doit pas être un fichier exécutable** (permissions x)
- Une fonction se termine soit après l'exécution de la dernière commande située avant l'accolade fermante, auquel cas **le code de retour** est celui de cette dernière commande, soit après exécution d'une **commande return n**, avec "n" un nombre compris entre 1 et 255.
- A l'intérieur de la fonction, il peut y avoir une ou plusieurs commandes « **return n** »
- Pour **utiliser une fonction** dans un script, il faut
  - Appeler le fichier contenant la (ou les) fonction(s) (avant de l'utiliser) : «**. ficFonctions** »
  - Utiliser la fonction comme une commande : « nomFonction »
  - Si nécessaire et prévu dans la fonction, lui passer des arguments : « nomFonction valeur1 valeur2… »

- Saisissez et testez le script suivant calcul.sh.

Calcul.sh	**Améliorez-le en ajoutant l'affichage de la moyenne :**

Function calcul()
{
let somme=$1+$2
}
calcul 5 9
echo $somme	

- **Ecrivez une fonction qui calcule la somme et la moyenne pouvant accepter jusqu'à 6 paramètres.**