12 SHELL SCRIPT

Remarques :

1) On peut utiliser toutes les commandes fournies dans le système dans le code (*ls, rm*, etc.).

2) Quand on fait du scripting, on met le chemin absolu et non le chemin relatif car le script peut se lancer à partir de différents endroits. Ex : /home/debian/TP/fichier.txt et non fichier.txt

1 VÉRIFIER UN SCRIPT

• https://www.shellcheck.net/

2 CRÉER ET EXÉCUTER UN SCRIPT

• **Créer un script :** nano fichier.sh

• **Exécuter un script :** bash fichier.sh argument1 argument2

OU : chmod u+x fichier.sh

./fichier.sh argument1 argument2

3 RÉDIGER LE SCRIPT (LES BASES)

• **Shebang qui indiquer quel interpréteur de commandes doit être utilisé pour exécuter le script :** #! /bin/bash

• **Faire un commentaire : #** commentaire

• **Afficher un message dans la console :** echo "message"

4 ARGUMENTS

• **Utiliser un argument dans le script :** $numéro_argument. Ex : argument1 -> $1

• **Récupérer le nom du script :** $0

• **Récupérer la chaine complète des arguments :** $*

• **Récupérer la liste des arguments tels qu'ils ont été fournis en ligne de commande :** $@

• **Récupérer le nombre d'arguments :** $#

• **Récupérer les options en ligne de commande activées pour le script en cours d'exécution :** $-

• **Récupérer le code de retour de la dernière commande exécutée** (retourne 0 si elle s'est exécutée avec succès, sinon 1) **:** $?

• **Récupérer le PID de processus du script en cours d'exécution :** $$

• **Récupérer le PID de processus en arrière-plan lancé dans le script :** $!

• **Dernier argument de la commande précédente exécutée :** $_

5 VARIABLES

• **Déclarer une variable :** variable="valeur"

• **Afficher la valeur d'une variable :** echo "${variable}" # valeur

'${variable}' # $variable

La commande est exécutée et le résultat est affiché : "`commande`"

• **Permettre l'interprétation des caractères d'échappement spéciaux :** echo -e "texte\n"

• **Demander à l'utilisateur de saisir une valeur et l'enregistrer dans une variable :** read -p "demande" variable

• **Vérifier si une variable est définie** (retourne 1 si elle est définie, sinon 0) **:** ${variable:+1}

• **Connaitre la longueur d'une variable :** ${#variable}

• **Récupérer une sous-chaine d'une variable :** ${variable:position:longueur}

• **Supprimer le plus petit préfixe de la variable qui correspond au pattern :** ${variable#pattern}

• **Supprimer le plus grand préfixe de la variable qui correspond au pattern :** ${variable##pattern}

• **Supprimer le plus petit suffixe de la variable qui correspond au pattern :** ${variable%pattern}

• **Supprimer le plus grand suffixe de la variable qui correspond au pattern :** ${variable%%pattern}

• **Afficher toutes les variables d'environnement :** env (les variables d'environnement sont définies dans le fichier « ~/.bashrc »)

• **Créer une variable d'environnement permanente :** export variable="valeur"

• **Créer une variable d'environnement temporaire :** set variable="valeur"

• **Détruire une variable d'environnement :** unset variable

6 INTERACTION UTILISATEUR

• **Permettre à l'utilisateur de sélectionner une option proposée :**

```
select variable in option1 optionN
do
  instructions
  exit
done
```

• **Affecter la valeur rentrée par l'utilisateur à une(des) variable(s) :**
read -p "demande" variable1 variableN

• **Affecter la valeur rentrée par l'utilisateur à une(des) variable(s), limiter le nombre de caractères lus à partir de l'entrée utilisateur et le temps autorisé (en secondes pour saisir la valeur et masquer le texte saisi :** read -p -n nombre -t secondes -s "demande" variable

7 TABLEAUX

• **Créer un tableau indexé :**

```
declare -a tableau
tableau=("valeur1" "valeur2" "valeur3")
```

• **Créer un tableau associatif :**

```
declare -A tableau
tableau=([un]="valeur1" [deux]="valeur2" [trois]="valeur3")
```

• **Récupérer un élément d'un tableau :** ${tableau[position]}, ${tableau[deux]}

• **Récupérer tous les éléments d'un tableau :** ${tableau[@]}

• **Récupérer le nombre d'éléments contenus dans le tableau :** ${#[tableau][@]}

8 OPÉRATEURS

• **Utiliser un opérateur simple dans une condition :** if [[opérateur variable/fichier/"chaine"]]

• **Utiliser un opérateur complexe dans une condition :**
if [[variable1/condition2 opérateur variable2/condition2]]

8.1 Opérateurs arithmétiques

• **Utiliser un opérateur arithmétique :** if ((variable1 opérateur variable2 == variable3))

• **+ :** addition

• **- :** soustraction

• *** :** multiplication

• **/ :** division

• **% :** modulo (obtenir le reste d'une division)

• **** :** exponentiation (élever un nombre à une puissance)

8.2 Opérateurs de comparaison numérique complexes

• **-eq =** égal

• **-ne =** non-égal

• **-lt =** plus petit que

• **-le =** plus petit ou égal

• **-gt =** plus grand que

• **-ge =** plus grand ou égal

8.3 Opérateurs de chaines de caractères

• **Opérateurs de chaines simple :**

— **-z =** vide

— **-n =** non vide

• **Opérateurs de chaines complexes :**

— **== :** égal

— **!= :** non-égal

— **\< :** inférieur à (dans l'ordre alphabétique)

— **\<= :** inférieur ou égal à (dans l'ordre alphabétique)

— **\> :** supérieur à (dans l'ordre alphabétique)

— **\>= :** supérieur ou égal à (dans l'ordre alphabétique)

8.4 Opérateurs de fichiers

• **Opérateurs de fichiers simples :**

— **-f :** le fichier est de type standard (c'est-à-dire un fichier régulier)

— **-d :** le fichier est de type répertoire

— **-r :** l'utilisateur possède les droits de lecture sur le fichier

— **-w :** l'utilisateur possède les droits d'écriture sur le fichier

— **-x :** l'utilisateur possède les droits d'exécution sur le fichier

— **-e :** le fichier existe

— **-s :** le fichier a une taille supérieure à zéro (c'est-à-dire s'il n'est pas vide)

• **Opérateurs de fichiers complexes :**

— **-nt :** le fichier1 est plus récent que le fichier2

— **-ot :** le fichier1 est plus ancien que le fichier2

— **-ef :** le fichier1 est un lien symbolique qui pointe vers le même fichier que le fichier2

8.5 Opérateurs logiques

• **Utiliser l'opérateur logique « ! » :** if ! [[condition]]

• **&&** ou **-a :** les deux conditions sont vraies

• **||** ou **-o :** au moins l'une des conditions est vraie

• **! :** inverse le résultat de la condition

9 CONDITIONS

• **if :**

```
if [[ condition ]];
then
  instructions
elif [[ condition ]];
then
  instructions
else
  instructions
fi
```

10 BOUCLES

• **Boucle for :**

```
for variable in $(seq 0 10)
do
  echo "$variable"
  instructions
done
```

OU : `for ((variable=0; variable<11 ; variable++))`

OU : `for variable in "${tableau[@]}"`

OU : `for variable in 1 2 3 4 5 6 7 8 9 10`

• **Boucle while :**

```
$variable = 0
while [[ condition ]]
do
  instructions
  ((variable++))
done
```

• **Boucle infinie :**

```
while [[ true ]]
do
  instructions
done
```

• **Boucle until** (jusqu'à ce que la condition soit vraie) :

```
until [[ condition ]]
do
  instructions
done
```

• **Arrêter une boucle :** break

11 SWITCH CASE

• **Case :**

```
case $choix in
  valeur1a | valeur1b) instruction;;
  valeur2) instruction;;
  *) instruction;;
esac
```

12 OPÉRATIONS ARITHMÉTIQUES

• résultat=$(($variable1+$variable2)) = let résultat=variable1+variable2 = résultat=$(expr $variable1+$variable2)

• résultat=$((1+3))

13 DÉCALAGE DE PARAMÈTRES

• **Décaler les arguments d'une position vers la gauche :** shift (l'argument $2 devient $1, etc.)

• **Décaler les arguments d'un nombre de positions vers la gauche :** shift nombre

14 FONCTIONS

• Remarque : les variables qu'une fonction utilise sont globales par défaut.

```
function nom_fonction() {
  instructions
}
```

• **Faire appel à la fonction :** fonction argument1 argument2

• **Créer une variable locale :** local variable=valeur

15 AUTRE

• **Arrêter le programme :** exit

• **Générer un nombre aléatoire entre 1 et 50 :** nombre=$(($RANDOM % 50 + 1))

• **Récupérer la date actuelle :** date_actuelle=$(date +%d-%m-%Y)