Scripting Shell GNU/Linux

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 : **03-ScriptingShell**

1. Quelques astuces

1.1. Quelques raccourcis utiles

Quelques séquences de raccourcis de commandes sont à connaître :

- [Ctrl] C : interruption du programme : il se termine.
- [Ctrl] Z : stoppe le programme (voir les processus).
- [Ctrl] D : interrompt une saisie sur un prompt >.

1.2. historique

En plus des flèches haut et bas qui permettent de naviguer dans l'historique, le shell permet l'accès au fichier caché .bash_history qui contient les dernières commandes lancées.

Vous pouvez voir le contenu de l'historique avec la commande history (qui affiche tout l'historique » ou la commande « fc –l » qui affiche les 15 dernières lignes.

Testez ces commandes.

2. Les Variables

On en distingue principalement deux types : variables utilisateur et variables système.

Le principe est de pouvoir affecter un contenu à un nom de variable, généralement une chaîne de caractère ou des valeurs numériques.

Une variable est déclarée dès qu'une valeur lui est affectée. **L'affectation est effectuée avec le signe =, sans espace avant ou après le signe**.

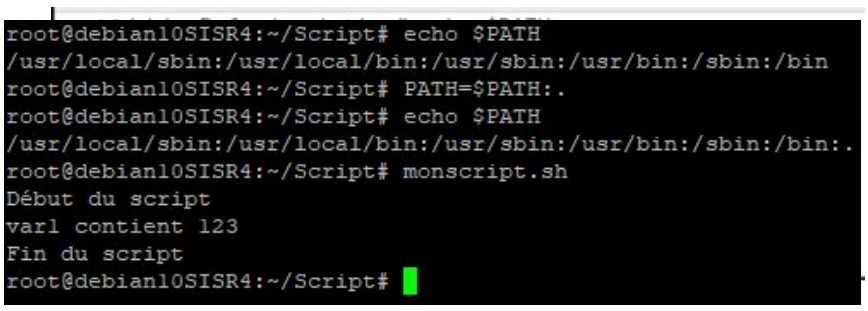
Vous accédez au contenu d'une variable en plaçant le signe **$** devant le nom de la variable. Quand le shell rencontre le **$**, il tente d'interpréter le mot suivant comme étant une variable. Si elle existe, alors le $nom_variable est remplacé par son contenu, ou par un texte vide dans le cas contraire.

2.1. La variable système PATH

A chaque **lancement du script**, il a fallu préciser devant son nom « **./** » pour indiquer que le script se trouve dans le répertoire de travail.

Si on omet les caractères « **./** », le shell cherchera ce script dans les répertoires contenus dans la **variable PATH**.

- Ajoutez dans cette variable le répertoire courant (représenté par « . ») **comme ci-contre.**



La commande importante est « PATH=$PATH:. » : elle redéfinie la variable PATH en prenant sa valeur et en ajoutant les caractères « :. » qui représentent le séparateur et le répertoire courant.

Pour conserver cette modification de la variable PATH, il faut l'inclure dans le fichier « .bashrc » …

2.2. Différence entre les guillemets (" ") et les apostrophes (' ')

Testez les exemples suivants et notez le résultat de l'instruction :

Instruction	Résultat
a=Jules	
b=Cesar	
c="$a $b a conquis la Gaule"	
echo $c	
d='$a $b a conquis la Gaule'	
echo $d	
echo "Linux c'est top"	
echo 'Linux " trop bien " '	

Conclusion, quelle est la différence entre les guillemets et l'apostrophe ?

2.3. Suppression et protection

Vous supprimez une variable avec la commande unset. Vous protégez une variable en écriture et contre sa suppression avec la commande readonly. Une variable en lecture seule, même vide, est figée. Il n'existe aucun moyen de la replacer en écriture et de la supprimer, sauf en quittant le shell.

Testez les exemples suivants et notez le résultat de l'instruction :

Instruction	Résultat
a=Jules	
b=Cesar	
echo $a $b	
unset b	
echo $a $b	
readonly a	
a=Neron	
unset a	

2.4. Calculs et variables typées

Les variables peuvent être typées en entier (integer) avec la commande typeset –i. L'avantage est qu'il devient possible d'effectuer des calculs et des comparaisons sans passer par les commandes « expr » « let » ou « ((…)) » qui permettent des calculs sur variables.

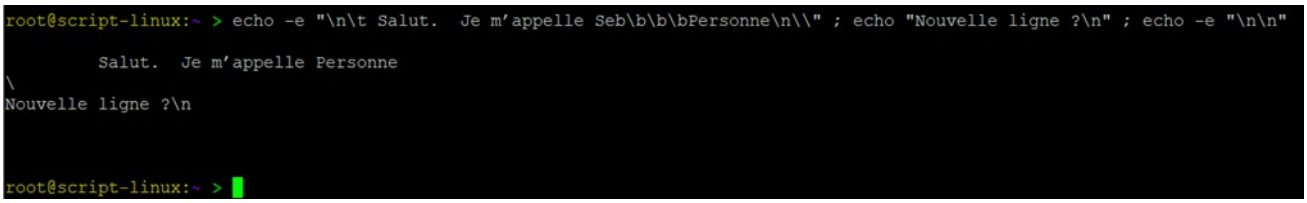
Opérateur	Rôle
+ - * /	Opérations simples
%	Modulo
< > <= >=	Comparaisons, 1 si vraie, 0 si faux
== !=	Égal ou différent
&& ||	Comparaisons liées par un opérateur logique
& | ^	Logique binaire AND OR XOR

Testez les exemples suivants et notez et commentez le résultat de l'instruction :

Instruction	Résultat
let resultat=5*5 echo $resultat	
resultat=5*5 echo $resultat	
typeset –i resultat resultat=Erreur echo $resultat	
resultat=6*7 echo $resultat	
resultat=resultat*3 echo $resultat	
typeset –i add add=5 resultat=add+5 resultat=resultat*add echo $resultat	
A=12 ; B=5 C=$((A*B)) Echo $C	

2.5. Afficher et mettre en forme du texte avec echo

Testez la suite de commandes « $ echo -e "\n\t Salut. Je m'appelle Seb\b\b\bPersonne\n\\" ; echo "Nouvelle ligne ?\n" ; echo -e "\n\n" » et complétez le tableau suivant.



	Description
-e	Permet d'interpréter les caractères suivants dé-spécifiés par un backslash (contre-oblique)
\n	???
\t	???
\b	Retour d'un caractère en arrière
****	Afficher le backslash (barre oblique inverse)

3. Structure et exécution d'un script

3.1. Commandes internes et externes

Il existe deux types de commandes :

- **Les commandes internes** sont internes au shell et exécutées au sein de celui-ci. Ces commandes font partie du programme shell, le bash. Les commandes cd ou pwd sont des commandes internes. Quand vous les exécutez, le shell exécute les fonctions définies en son sein correspondant à celles-ci.

- **Les commandes externes** sont des programmes binaires présents en tant que fichiers sur votre disque dur (ou tout autre support de données). Quand vous exécutez la commande, ce fichier est chargé en mémoire et lancé en tant que processus.

Vous pouvez distinguer une commande interne d'une commande externe à l'aide de la commande interne **« type » suivi du nom de la commande**.

Quels sont les types des commandes « date » et « pwd » ?

3.2. Notions de bases

Le shell dispose d'un véritable langage de programmation avec notamment une gestion des **variables**, des tests (**alternatives**) et des boucles (**itératives**), des opérations sur les variables, des fonctions…

Lors de son exécution d'un script, chaque ligne sera lue une à une et exécutée.

Une ligne peut se composer de commandes internes ou externes, de commentaires ou être vide. Plusieurs instructions par lignes sont possibles, séparées par le **;** ou liées conditionnellement par **&&** ou **||**.

Le « **;** » est l'équivalent d'un **saut de ligne**.

Par convention les noms des shell scripts se terminent généralement (pas obligatoirement) par « **.sh** » pour le **Bourne Shell** et le **Bourne Again Shell, par « .ksh »** pour le **Korn Shell** et par « **.csh** » pour le **C Shell**.

3.1. Création d'un script avec l'opérateur de redirection

Il y a plusieurs façons de créer un script. Le plus souvent, nous utilisons un éditeur, mais il est possible de le faire en ligne de commande avec la commande « echo » et l'opérateur de redirection.

- S'il n'existe pas, Créer un **répertoire « scripting »** dans le répertoire par défaut de root
- Déplacez-vous dans ce répertoire
- Tapez les commandes suivantes (Attention après chaque commande, il y a un espace !)
-

```
# touch monscript.sh              # Création d'un fichier nommé monscript.sh vide
# echo "# Ceci est une ligne de commentaire" > monscript.sh
# echo "echo 'Début du script'" >> monscript.sh          # Ecriture d'une 1ère commande dans le script
# echo "echo 'Fin du script'" >> monscript.sh     # Ajout d'une 2nde commande dans le script
# ls -l                           # Par défaut, ses droits sont "-rw-r--r--", donc pas exécutable
# chmod +x monscript.sh
# ./monscript.sh                  # Exécution du script
```

Explications :

- Le caractère « **#** » permet de placer des commentaires dans un script. Tout ce qui suit ce caractère ne sera pas interprété par le shell.
- La commande « **touch monscript.sh**» permet de créer un fichier nommé monscript.sh vide (ou modifier l'horodatage s'il existe)
- La commande « **echo** "Bonjour" » qui affiche tout ce qui se trouve entre les deux délimiteurs " … "
- La **redirection** qui permet de rediriger le résultat de la commande echo dans le fichier monscript.sh (opérateur **>**) ou d'y ajouter de nouvelles lignes (opérateur **>>**).

4. Ecriture d'un script utile

Nous allons écrire un script qui nous sera utile tout au long de ce TP.

Pour créer un script, il faut :

1. Créer un fichier avec une extension .sh
2. Mettre en première ligne le bash à utiliser, pour nous : #!/bin/bash
3. Modifier les permissions pour le rendre exécutable
4. Taper le script

Vous allez créer un script qui va automatiser les 3 premières actions, équivalentes en ligne de commande à :

1. « **touch nomFichier.sh** » permet de créer le script
2. « **echo "#!/bin/bash" > nomFichier.sh** » permet de mettre la première ligne
3. « **chmod u+x nomFichier.sh** » permet de le rendre exécutable

➢ **Créez un fichier nommé e.sh** qui contient les lignes suivantes (sans les numéros !) (Pourquoi e.sh ? e comme exécutable et simple à saisir lorsqu'on utilise la complétion automatique e. +<tab>)

Voici le contenu :

```
1 #!/bin/bash
2 touch $1.sh
3 echo "#!/bin/bash" > $1.sh
4 chmod u+x $1.sh
```

Explications des lignes :

1. Tous les scripts commencent par préciser le shell utilisé, ici le bash
2. Création du fichier dont le nom est passé en argument (**$1**) avec l'extension **.sh**
3. Insertion de la 1ère ligne d'un fichier de script shell
4. Positionnement du droit d'exécution sur ce fichier

Dans la suite de cette formation, lorsque vous aurez à créer un nouveau script, par exemple test.sh, il suffira de taper la commande suivante :

```
e.sh test                # Crée le fichier test.sh : à vérifier et tester
```

➢ Pour pourvoir l'utiliser, il est nécessaire **d'exécuter** avant la commande « **chmod +x e.sh** »

5. Quelques astuces (2)

5.1. Personnaliser la page d'accueil *** Facultatif ***

Personnaliser la page d'accueil permet de rajouter un visuel à la connexion de l'utilisateur. Cela permet de ne pas se tromper de serveur.

Les informations affichées à la connexion d'un utilisateur sont stockées dans le fichier /etc/motd

Pour générer un texte en ASCII ART, utiliser le site : http://patorjk.com/software/ ou les commandes toilet ou figlet ; vous pouvez aussi utiliser fortune et/ou cowsay.

Une bibliothèque est aussi présente ici https://www.asciiart.eu/ .

5.2. Personnalisation du prompt *** Facultatif ***

Le prompt contient par défaut votre nom d'utilisateur et le nom de la machine. Il est définit dans la variable PS1 dans le fichier « .bashrc » des utilisateurs.

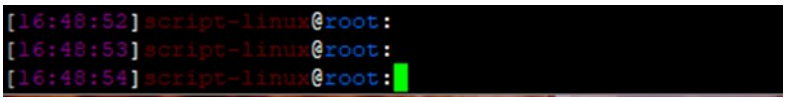
Il faut mettre la couleur entre [] et le paramètre, par exemple : \[\e[38;5;90m\]\t]:

Pour simplifier la lecture du paramètre, les couleurs peuvent être passées en variables.

Voici la liste des quelques paramètres utilisables :

- \u nom de l'utilisateur courant
- \h nom de la machine
- \W nom du répertoire courant
- \$ privilège de l'utilisateur courant
- \d date courante (au format lun. janv. 1)
- \w chemin complet du répertoire de travail
- \A heure format 24h
- \D {%d-%m-%Y %H:%M:%S%z}Date et heure dans un format personnalisable
- \j nombre de tâches en cours dans le terminal
- \# numéro de la commande dans l'historique
- \v version de bash

Il existe un site qui vous aide dans la personnalisation du bashrc : https://itsfoss.com/bash-prompt-generator/ .



6. Les paramètres d'un script

6.1. Paramètres de position

Les **paramètres de position** sont aussi des variables spéciales utilisées lors d'un **passage de paramètres** à un script.

Variable	Contenu
$0	Nom de la commande (du script).
$1-9	$1,$2,$3… Les neuf premiers paramètres passés au script.
$#	Nombre total de paramètres passés au script.
$*	Liste de tous les paramètres au format « $1 $2 $3 … ».
$@	Liste des paramètres sous forme d'éléments distincts « $1 » « $2 » « $3 » …

- Saisissez et testez le script **param.sh** en l'appelant avec les arguments suivants : # **param.sh riri fifi loulou** et identifiez les paramètres $0, $#, …

```
#!/bin/bash
echo "Nom : $0"
echo "Nombre de parametres : $#"
echo "Parametres : 1=$1 2=$2 3=$3"
echo "Liste : $*"
echo "Elements : $@"
```

Variable	**1. Complétez** le résultat
$0	Nom :
$#	Nombre de parametres :
$1, $2, $3	Parametres :
$*	Liste :
$@	Elements :

6.2. Redéfinition des paramètres

L'instruction **set** permet de **redéfinir le contenu des variables de position** pour par exemple affecter des valeurs par défaut lorsque le passage de paramètre est facultatif.

Avec : **set valeur1 valeur2 valeur3 …,** $1 prendra comme contenu valeur1, $2 valeur2 et ainsi de suite.

2. Saisissez le script et testez-le avec les paramètres suivants : **# param2.sh riri fifi loulou**

param2.sh	Notez, **complétez** le résultat
#!/bin/bash echo "Avant :" echo "Nombre de parametres : $#" echo "Parametres : 1=$1 2=$2 3=$3 4=$4" echo "Liste : $*" set alfred oscar romeo zoulou echo "apres set alfred oscar romeo zoulou" echo "Nombre de parametres : $#" echo "Parametres : 1=$1 2=$2 3=$3 4=$4" echo "Liste : $*"	Avant : Nombre de parametres : Parametres : Liste : apres set alfred oscar romeo zoulou Nombre de parametres : Parametres : Liste :