Initiation aux Scripts: Bash, PowerShell et Python

Initiation aux Scripts Bash Linux

Initiation aux Scripts Bash Linux

- Le **Bourne Again Shell (Bash)** est une interface en ligne de commande, et est le Shell par défaut dans la majorité des distributions Linux. Il permet d'**exécuter des commandes**, **d'automatiser des tâches** via des **scripts**, et d'explorer de nombreuses possibilités de **programmation**. Ce qui le rend unique, c'est que la ligne de commande elle-même est un programme appelé **Shell**, qui interprète les instructions.

Bash est un Shell, mais **tous les Shells ne sont pas Bash**.

#!/bin/bash

Script Bash : Création et exécution

- Créer un fichier script1.sh , avec n'importe quel éditeur (nano edito):
  - $ nano script1.sh
- Ainsi, pour lancer la commande hello world par exemple, on ajoute le script suivant puis on enregistre le fichier.

```
#!/bin/bash
echo "Hello World"
```

- Pour passer à l'exécution de ce fichier sur Bash, il existe deux manières:

➢ La première consiste à exécuter directement le fichier dans Bash comme ceci :

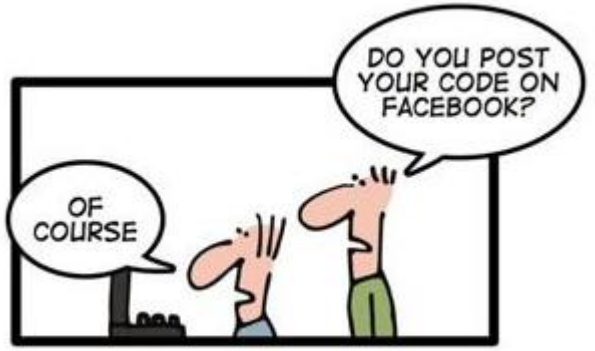
- bash script1.sh

➢ Et la deuxième façon consiste à demander l'autorisation de l'exécution sur Bash puis passer à l'exécution. Cela se révèle comme ceci :

- chmod a+x script1.sh
- ./script1.sh

- Une fois que l'exécution a été faite sur Bash, la sortie se présente comme ceci : Hello world

Script Bash : Commentaires

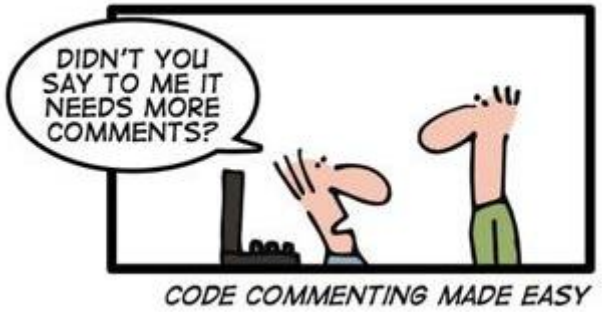


- Dans le scripting Bash, le symbole « # » est utilisé pour obtenir un commentaire sur une seule ligne. Pour ce faire, créez un fichier appelé « commentaire_exp.sh ». Ensuite, ajoutez le script suivant sur une seule ligne de commentaire :

```
#!/bin/bash
# Ajouter deux valeurs numeriques ((som= 45 + 15 ))
#Afficher le resultat
echo $som
```

- **Commentaire multi-ligne**
- Sur Bash, vous pouvez aussi exécuter un commentaire sur plusieurs lignes. « comment_multiligne.sh. Ensuite, pour exécuter un commentaire multiligne, vous devez utiliser les symboles « : » ou/et « ‘ ». Dans ce script, il est demandé de calculer le carré du chiffre 9.

```
#!/bin/bash
: '
Le script suivant, permet de
calculer le carré du chiffre 9.
'
((carre=9*9))
echo $carre
```



Script Bash : Les variables

Les variables d'environnement et les variables de shell ont chacune leur propre portée. Lorsque vous exportez une variable par exemple,

- **export** MY_ENV_VAR="var1"

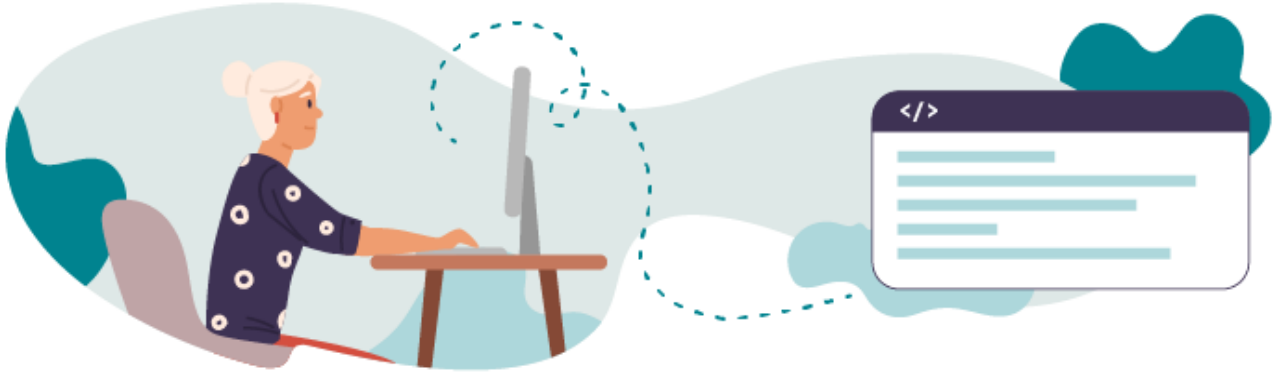
elle devient disponible pour tout script/sous-processus lancé à partir de ce shell.

- Une variable de shell régulière n'est visible que dans la session actuelle.
- Une variable exportée est disponible pour les processus enfants lancés à partir de cette session.
- Une variable définie dans un fichier de démarrage de shell (comme .zshrc) est appliquée à toutes les nouvelles sessions de ce shell.

Script Bash : L’entrée de l’utilisateur

- Dans bash, on utiliser la commande « read » pour obtenir l’entrée d’un utilisateur.
- Dans l’exemple ci-dessous, on créer un fichier appelé user_input.sh. la valeur de chaîne sera prise en combinaison avec une autre variable de chaîne et affichera ensuite les résultats. Ajouter le script suivant au fichier :

```
#!/bin/bash
echo "Entrer Votre Nom"
read nom
echo "Bonjour $nom a Linux "
```



- Afficher un message avant la saisie: read -p "Entrez votre nom: " nom

Script Bash : La boucle WHILE

- La boucle while permet d'exécuter **répétitivement** un bloc de commandes **tant qu'une condition est vraie**.

```
while [ condition ]; do
# commandes à répéter
done
```

- Exemple: Script Bash utilise une boucle while pour afficher la valeur de i tant qu'elle est inférieure ou égale à 3, en l'incrémentant (ajoutant 1) à chaque tour.

```
#!/bin/bash
i=1
while [ $i -le 3 ]; do
  echo "i vaut : $i"
  ((i++))
done
```

Opérateur : Signification

-eq	: égal à
-ne	: différent de
-lt	: inférieur à
-le	: inférieur ou égal à
-gt	: supérieur à
-ge	: supérieur ou égal à

Script Bash : L’instruction if

- La structure if en Bash permet d'exécuter un bloc de commandes seulement si une condition est vraie. Elle s'utilise pour prendre des décisions dans un script.

```
if [ condition ]; then
# commandes si la condition est vraie
fi
```

- Exemple: Script Bash qui vérifie si la variable i est inférieure ou égale à 3, et affiche un message si c’est le cas.

```
#!/bin/bash
i=2
if [ $i -le 3 ]; then
  echo "i est inférieur ou égal à 3"
fi
```

Opérateur : Signification

-eq	: égal à
-ne	: différent de
-lt	: inférieur à
-le	: inférieur ou égal à
-gt	: supérieur à
-ge	: supérieur ou égal à

Script Bash : Avec Python

- Installer python:
  - sudo apt-get update
  - sudo apt-get install python3
- vérifier la version installée avec
  - python --version
- Ecrire le script par exemple script2.py

```
#!/usr/bin/env python3
print("Bonjour depuis Python !")
```

- Exécuter le script avec l'interpréteur Python:
  - **python3 script.py**
- Ou depuis le Shell Bash:
  - chmod a+x script2.py
  - ./script2.py

Initiation Script sous PowerShell

Initiation Script sous PowerShell

L'invite de commande (CMD) et PowerShell facilitent l’interaction en ligne de commande, mais avec des différences significatives en termes d’approche, de capacités et de public cible:

- CMD, fournit une interface textuelle de base pour la gestion du système.
- À l’inverse, PowerShell introduit une approche moderne et dynamique des scripts et de l’administration en ligne de commande.

Script PowerShell : Python

- Un script Python lancé depuis PowerShell permet d'exécuter du code Python dans l'environnement Windows. PowerShell agit comme un intermédiaire il appelle l'interpréteur Python pour exécuter un fichier script3.py dans l'exemple qui suit.
- Pour installer python sous windows
  - Menu démarrer->microsoft store-> python3.13
- Pour vérifier la version de python installée sous PowerShell
  - Python --version

Script PowerShell : Python

- Un script Python lancé depuis PowerShell permet d'exécuter du code Python dans l'environnement Windows. PowerShell agit comme un intermédiaire : il appelle l'interpréteur Python pour exécuter un fichier script3.py dans l'exemple qui suit.
- Par exemple, si script3.py contient :
  - print("Bonjour depuis Python !")
- Utilise PowerShell pour lancer et exécuter le script script3.py avec :
  - python3 .\script3.py ou python3 script3.py
- Alors PowerShell affichera :
  - Bonjour depuis Python !

TP1:

a) Écris un script Bash qui demande à l'utilisateur d'entrer un nombre, puis affiche :

- "Petit" si le nombre est inférieur à 10
- "Exactement 10" si le nombre est égal à 10
- "Grand" si le nombre est supérieur à 10

Opérateur de chaine	**: Signification**
=	: égal à
!=	: différent de

b) Écris un script Python qui demande à l'utilisateur de taper un mot de passe

- Tant que le mot de passe n'est pas "secret", il redemande
- Quand le mot de passe est correct, il affiche "Accès autorisé"

TP 2

- Vous devriez être dans le répertoire /home/cnn/project
- Créer une variable nommée my_var et lui assigner une valeur
  - ma_var="Bonjour, Linux"
- Afficher la valeur de la variable avec: echo $ma_var
  - echo “La valeur ma_var is: $ma_var“
- Afficher toutes les variables d'environnement actuelles avec: env
- Afficher la variable d’environnement PATH : echo $PATH
- Utiliser export pour créer une variable d'environnement
  - export MA_VAR_ENV=“Ceci est une variable d’environnement“
- Créer un script shell qui tente d'accéder à une variable de shell régulière et à une variable d'environnement.
- cat test_vars.sh

```
#!/bin/bash
echo "Shell variable: $my_var"
echo "Environment variable: $MY_ENV_VAR"
```

- chmod +x test_vars.sh
- Exécuter le script avec: ./test_vars.sh
- Qu’affiche et expliquer pourquoi

TP3 :Ecriture des règles udev dans script bash

- Identifier ton périphérique USB en utilisant:
  - lsbk
  - lsblk -o NAME,SIZE,VENDOR,MODEL
  - udevadm info -n /dev/sdb1 # si c'est un périphérique de stockage
- Créer une règle personnalisée :
  - sudo nano /etc/udev/rules.d/99-usb.rules
- Contenu possible de la règle :
  - SUBSYSTEM=="usb", ATTR{idVendor}=="YYYY", ATTR{idProduct}=="ZZZZ", ACTION=="add", RUN+="/usr/local/bin/usb-action.sh"
- Créer le script usb-action.sh avec: sudo nano /usr/local/bin/usb-action.sh

```
#!/bin/bash
echo "$(date): USB connecté" >> /var/log/usb-event.log
```

- Exécuter le script:
  - sudo chmod +x /usr/local/bin/usb-action.sh
  - sudo /usr/local/bin/usb-action.sh
- Tester le script : cat /var/log/usb-event.log
- Recharger les règles :
  - sudo udevadm control --reload-rules
  - sudo udevadm trigger