MEWO

Module 2 :

Créer des scripts shell simples

Timothee SICCHIA
11/10/2024

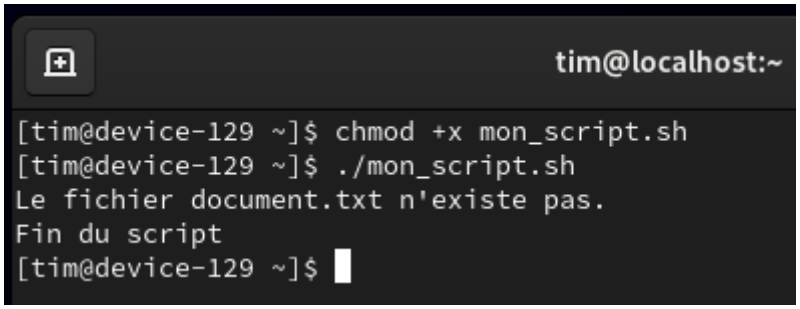
Table des matières

1. Exécuter du code conditionnel (utilisation de : if, test, [], etc.)

- **Explication** : Le code conditionnel permet d'exécuter des commandes spécifiques en fonction d'une condition. En shell, on utilise souvent les instructions `if` accompagnées de `test` ou [ ] pour évaluer ces conditions.
- ***Exemple*** : Écrivons un script qui vérifie si un fichier existe.

```
#!/bin/bash

if [ -f "/home/tim/document.txt" ]; then

echo "Le fichier document.txt existe."

else

echo "Le fichier document.txt n'existe pas."

Fi

echo "Fin du script"
```

- **Explication de l'exemple** : Ce script utilise `if` pour vérifier l'existence de `document.txt` dans le dossier `/home/tim`. L'option `-f` indique si le fichier existe et est un fichier régulier (fichier standard qui contient des données, du texte ou du code (par opposition à un répertoire, un lien ou un périphérique)). Si la condition est vraie, le script affiche un message ; sinon, il affiche un message alternatif (condition « sinon »). Cela permet de conditionner l'exécution des commandes en fonction de la présence d'un fichier, d'un répertoire ou d'un autre élément.



Ne pas oublier de rendre le script exécutable en gérer les permissions via chmod

2. Utiliser des boucles (for, etc.) pour traiter des fichiers et des entrées en ligne de commande

- **Explication** : Les boucles for permettent d'exécuter une série de commandes de manière répétitive, ce qui est très utile pour automatiser des tâches répétitives.
- ***Exemple*** : Écrivons un script qui affiche le nom de tous les fichiers dans un répertoire.

```
#!/bin/bash

for fichier in /home/tim/Documents/*; do

    echo "Traitement du fichier : $fichier"

done
```

- **Explication de l'exemple** : Ce script utilise une boucle `for` pour parcourir tous les fichiers dans le répertoire `/home/tim/Documents`. Pour chaque fichier, la commande `echo` affiche le chemin complet du fichier. Les boucles `for` sont couramment utilisées pour automatiser le traitement de plusieurs fichiers, ce qui est très pratique pour des opérations répétitives.

3. Gérer les entrées de script ($1, $2, etc.)

- **Explication** : Lorsqu'on exécute un script, on peut lui passer des arguments en ligne de commande. Ces arguments sont référencés dans le script par `$1`, `$2`, etc., ce qui permet de rendre le script plus flexible et interactif.
- ***Exemple*** : Écrivons un script qui copie un fichier d'un emplacement à un autre, en utilisant les arguments passés en ligne de commande.

```
#!/bin/bash

source_file=$1

destination=$2


if [ -f "$source_file" ]; then

    cp "$source_file" "$destination"

    echo "Le fichier $source_file a été copié vers $destination."

else

    echo "Le fichier source n'existe pas."

Fi
```

- **Explication de l'exemple** : Lors de l'exécution du script (`./mon_script.sh fichier.txt /home/tim/Destination/`), `$1` correspond à `fichier.txt` et `$2` au répertoire de destination (`/home/tim/Destination/`). Le script vérifie si le fichier source existe, puis le copie à l'emplacement indiqué. Cela permet de rendre les scripts plus génériques et réutilisables avec différents fichiers et destinations.

4. Traiter la sortie des commandes shell à l'intérieur d'un script

- **Explication** : Il est possible de capturer la sortie d'une commande shell et de l'utiliser dans le script, ce qui permet d'automatiser et de prendre des décisions basées sur cette sortie.
- ***Exemple*** : Écrivons un script qui compte le nombre de fichiers dans un répertoire et affiche ce nombre.

```
#!/bin/bash

dossier="/home/user/Documents"

nombre_fichiers=$(ls -1 "$dossier" | wc -l)
```

```
echo "Le dossier $dossier contient $nombre_fichiers fichiers."
```

Explication de l'exemple : Le script utilise `$(ls -1 "$dossier" | wc -l)` pour compter le nombre de fichiers dans le dossier spécifié. La commande `ls -1` liste les fichiers ligne par ligne, et `wc -l` compte le nombre de lignes. La valeur est stockée dans la variable `nombre_fichiers` et affichée. Cela montre comment utiliser efficacement les sorties de commandes dans un script pour obtenir des informations dynamiques.

Résultat :

1. `ls -1`

Si un répertoire contient trois fichiers, `ls -1` affichera :

```
fichier1.txt
fichier2.txt
fichier3.txt
```

2. `wc -l`

La commande `wc -l` compte le nombre de lignes dans la sortie fournie par la commande précédente `ls -1`. Dans ce cas, elle prend en entrée la sortie de `ls -1` (la liste des fichiers) et compte le nombre de lignes. Si `ls -1` affiche trois fichiers, `wc -l` renverra :

```
3
```

Dans le script, cela permet de compter efficacement combien de fichiers sont présents dans le répertoire spécifié.