Les scripts Bash

Partie 3 - Structuration du code

Sommaire

Au menu :

01 Les fonctions

02 Usages et bonnes pratiques

Les fonctions

Fonction

Définition

Une **fonction** est un bloc de code nommé qu'on déclare pour pouvoir l'utiliser plus tard, éventuellement plusieurs fois.
Elles permettent de structurer son code, de favoriser sa réutilisation, sa maintenance, etc.

Attention : les fonctions doivent être déclarées avant d'être appelées, donc en général placée en début de script.

Déclarer une fonction

Ma première fonction

Syntaxe 1 :

```
function nom()
{
        instructions
}
```

Syntaxe 2 (sans **function**) :

```
nom()
{
        instructions
}
```

```
#!/bin/bash

# Declaring the hello function
function hello()
{
        echo "Hi folks !"
}
hello
echo "and again"
hello

wilder@host:~$ ./script.sh
Hi folks !
and again
Hi folks !
```

Fonctions et paramètres

Des fonctions adaptables

Un appel de fonction peut être suivi **d'arguments**.

On les récupère dans la fonction comme les **paramètres** d'un script.

```
#!/bin/bash
function hello() { echo "Hi folks !";}
greet()
{
        if [ $# -gt 0 ]
        then
                echo "Hi $1"
        else
                hello
        fi
}
greet wilder
greet
```

Valeur de retour

Définition

2 façon de sortir une valeur depuis une fonction :

- **return <code>** : envoi un code de sortie numérique entre 0 et 255
- **echo <valeur texte>** : envoi une valeur texte

```
#!/bin/bash

fonc_calcul()
{
        somme=$(($1 + $2))
        echo $somme
}
echo "Donne 2 nombres"
read -p "Le premier ? " nbr1
read -p "le second ? " nbr2
resultat=$(fonc_calcul $nbr1 $nbr2)
echo "Le résultat de la somme de
$nbr1 et $nbr2 est $resultat"
exit 0
```

Périmètre

Définition

Par défaut, dans un script une variable est globale c.a.d. que sa valeur est connue dans l'ensemble du script.
Pour que la valeur reste dans la fonction où elle est déclarée, il faut utiliser le mot clé **local**.

Périmètre (suite)

Définition

Exécute les 2 scripts suivant pour voir la différence.

```
#!/bin/bash

test_fonc()
{
        var="Bonjour"
}
var="Au revoir"
test_fonc
echo "$var"
exit 0
```

```
#!/bin/bash

test_fonc()
{
        local var="Bonjour"
}
var="Au revoir"
test_fonc
echo "$var"
exit 0
```

Usages et bonnes pratiques

Mettre un shebang

Bien commencer

Cela permet de spécifier dès le début l'interpréteur de commandes qui est utilisé.

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

pour une meilleure portabilité :

```
#!/usr/bin/env bash
echo "Hello !"
```

Mettre des commentaires

Pour se rappeler

Cela permet d'expliquer ce que fait le code.

Ces commentaires sont pour les autres, mais surtout pour toi !

```
#!/bin/bash

# Mise à jour de la liste des paquets
apt update

# Mise à jour du système
 apt upgrade -y
```

Mettre des commentaires (suite)

Pour se rappeler

On peut même en mettre en en-tête de script :

```
#!/bin/bash

##########################################
# addUser.sh
# Utilité: ce script sert à créer des utilisateurs passés en argument
# Usage: addUser.sh utilisateur1 utilisateur2 …
# Auteur: John DOE <j.doe@wcs.com>
# Mise à jour le: 01/02/2025
##########################################
```

Ne pas mettre sudo dans les scripts

De la sécurité

Il vaut mieux exécuter un script avec **sudo** que mettre sudo dans le script :

- Plus de contrôle sur les permissions : l'utilisateur sait qu'il exécute un script avec sudo
- Meilleure sécurité : pas d'élévations de privilèges sur des parties du script non-essentielles
- On peut exécuter le script avec un utilisateur précis avec **sudo -u <utilisateur>**

Ne pas mettre sudo dans les scripts (suite)

De la sécurité

Bonne pratique :

```
sudo ./script.sh
```

Mauvaise pratique :

```
#!/bin/bash
# Mise à jour de la liste des paquets et du système
sudo apt update && sudo apt upgrade -y
```

Ne pas mettre sudo dans les scripts (suite)

De la sécurité

Bonus : on peut mettre une vérification en début de script

```
#!/bin/bash
# Vérification d'exécution en sudo
if [[ $EUID -ne 0 ]]
then
        echo "Exécution du script obligatoire en sudo" >&2
        exit 1
fi
# Mise à jour de la liste des paquets
apt update
exit 0
```

Utiliser des variables

Garder les informations

Cela permet de rendre le code plus flexible et modifiable.

```
./script.sh wilder 25
```

```
#!/bin/bash

nom="$1"
age="$2"

read -p "Métier : " metier

echo "$nom a $age ans et son métier est $metier"
```

Utiliser des variables (suite)

Garder les informations

Il peut être judicieux de mettre des double guillemets pour encadrer les variables pour éviter les espaces non-désirés

```
#!/bin/bash

nom="$1"
age="$2"

read -p "Métier : " metier

echo "$nom a $age ans et son métier est $metier"
```

Utiliser des variables (suite)

Sécuriser les informations

Mauvais code :

```
var="string with spaces"
[ $var = "string with spaces" ] && echo "OK" || echo "KO"
```

Bon code :

```
var="string with spaces"
[ "$var" = "string with spaces" ] && echo "OK" || echo "KO"
[[ $var = "string with spaces" ]] && echo "OK" || echo "KO"
```

Gérer les erreurs

Eviter les sorties de route

Erreurs de saisie, mauvais typage de variable, ... doivent être gérées pour éviter des sorties du code non-prévues.
Avec || :

```
cd dossier10 2>/dev/null && echo "Déplacement dans le dossier" || echo "Dossier inexistant"
```

La commande **case** permet aussi de faire des choix suivant des erreurs prévues et non-prévue.

Utiliser des fonctions

De la sécurité

En les utilisant, le code peut être plus lisible, et cela permet de gagner des lignes de codes en évitant les répétitions.

```
#!/bin/bash
# Fonction de calcul
calculer() {
        nombre1="$1"
        nombre2=”$2"
        resultat=$((a + b))
        echo "$resultat"
}

calculer 5 10
calculer 1 3
calculer 10 200
```

Utiliser des fichiers de log

Eviter les sorties de route

Cela permet de journaliser l'activité et de faciliter le débogage.

```
#!/bin/bash

cd dossier10 2>/dev/null && echo "Déplacement dans le dossier" || echo "Dossier inexistant"
```

La commande **case** permet aussi de faire des choix suivant des erreurs prévues et non-prévue.

Des applications/modules de vérification

Se faire aider

- ShellCheck
- shfmt

Références

La doc officielle
Le wikibooks : Programmation Bash
Le Wiki Bash Hackers
Le Bash Guide de Greg
ExplainShell

Conclusion

Les fonctions
Retour sur les BP

Beaucoup de notions => Beaucoup de pratique

Ecrivez plein de scripts pour tout !!