Chapitre 6 - Les scripts shell bash

2.0

Système d'exploitation
GNU/Linux

GORGOUMACK SAMBE

1 Avril 2022

Table des matières

Objectifs

A l'issue de ce chapitre, l'apprenant doit être capable de

1. implémenter un script shell et de le lancer ;
2. utiliser des expressions et des variables sur bash;
3. utiliser des structures conditionnelles sur bash;
4. utiliser des structures itératives sur bash;
5. utiliser des fonctions sur bash.

Introduction

Les scripts shell permettent d'automatiser des tâches mais aussi de simplifier des commandes complexes. Ils sont interprétés par le shell et sont de simples fichiers texte composés de commandes, de variables et d'expressions, de structures de contrôle et itératives, de fonctions, ...

I.Les bases des scripts shell

1. Script shell

Plusieurs shell sont disponibles sur un système GNU/Linux, chaque shell possède **son propre langage** mais ils partagent tous **les même principes**.

Un **script shell** est un **simple fichier texte** permettant de **simplifier des commandes complexes** et d'**automatiser des tâches**.

2. Le shell bash

Dans ce cours nous utiliserons le **shell bash** qui est le **shell du projet GNU** et shell par défaut des systèmes **GNU/Linux**.

Bash est un **interpréteur de commandes** (shell) **compatible sh** qui exécute les commandes lues depuis l'entrée standard ou depuis un **fichier**. Bash inclut aussi des fonctionnalités utiles des interpréteurs de commandes **Korn** et **C** (ksh et csh). (tiré de la page de manuel de bash).

3. Édition et enregistrement d'un script shell

Un script shell est un **simple fichier texte**. Vous pouvez donc utiliser un **éditeur de texte simple** comme vi, vim, emacs ou encore gedit pour éditer votre fichier.

Il est recommandé de donner une **extension .sh** aux scripts shell.

4. Exécution d'un script shell

Un fichier créé dans un système GNU/Linux ne possède pas par défaut les droits d'exécution. Vous avez deux moyens pour exécuter un script shell bash :

1. Appeler la commande bash en lui passant en paramètre le nom de fichier :
   $ bash nom_script.sh
2. rendre le fichier exécutable :
   $ chmod u+x nom_script.sh
   pour pouvoir l'exécuter sans difficultés :
   $ ./nom_script.sh

Remarque : le point (./) indique que le fichier exécutable est dans le répertoire courant. Si vous ne le faites pas figurer, la commande (le script) sera recherché dans les chemins contenu dans le PATH. Si vous souhaitez l'exécuter sans ce point, il faudra ajouter le répertoire contenant le script à votre PATH.

5. Exemple de Hello world !

Exemple

Prenons un exemple avec Hello world !

1. Ouvrez l'éditeur de texte de votre choix e t saisissez le code en fin de section
2. enregistrer le fichier sous helloworld.sh
3. exécuter le fichier avec la commande :
   $bash helloworld.sh

```
1  #!/bin/bash
2  # helloworld.sh
3  # Affiche le message Bonjour tout le monde
4  echo Bonjour tout le monde
```

Vous pouvez également rendre le fichier exécutable puis l'exécuter.

II.La syntaxe du shell bash

Nous introduisons ici les éléments de base du langage. vous pouvez vous documenter beaucoup plus sur le sujet à travers la page de manuel de bash et la documentation gnu.

1. Shebang et commentaires

shebang

Le **shebang** désigne la séquence de caractères **#!** qui commence la **première ligne** d'un script et qui est **suivi par un nom d'interpréteur**. Elle sert à **indiquer** au système l'**interpréteur de script à utiliser** pour exécuter le code.

Voila deux exemples :

1. #!/bin/bash
2. #!/usr/bin/python

Le premier exemple pour un script à exécuter par bash et le deuxième pour un script à exécuter avec l'interpréteur python. Si on ne précise pas le shebang dans l'entête du script, il va être exécuté par l'interpréteur par défaut de l'utilisateur qui lance le script.

Commentaires

Un commentaire sur bash commence par le caractère #

2. Les variables

Déclaration de variables

On définit une variable en lui affectant une valeur :

mavar = valeur

mavar sera ici une variable de type chaîne de caractère.

Remarque

L'usage de variables chaînes de caractère n'est pas une contrainte pour beaucoup de programmeurs shells, il existe des moyens pour les traiter comme des valeurs numériques.

Certaines commandes comme **declare** permettent aussi de déclarer des variables typés , par exemple :

declare -i x

x sera une variable entière

Accès aux variables

pour accéder à la valeur d'une variable, il suffit de la préfixer par $:

echo $mavar

on peut marquer le nom d'une variable avec des accolades :

echo ${x}

3. Les tableaux

Types de tableau

bash permet d'utiliser des **tableaux indicés** et des **tableaux associatifs**. Les tableaux sur bash peuvent être des tableaux creux(on peut sauter des indices).

Création de tableaux

Quatre méthodes peuvent être utilisées pour créer un tableau :

1. Avec la commande **declare** et l'**option -a** :
   declare -a tableau
2. par **affectation** directe des valeurs sans indice (pour **un tableau indicé**) :
   tableau=('valeur0' 'valeur1' 'valeur2')
3. par **affectation** directe des valeurs avec indice (pour **un tableau associatif**) :
   tableau_asso=(['un']="valeur0" ['deux']="valeur1" ['trois']="valeur2")
4. par **lecture au clavier** avec la commande **read** :
   read -a tableau

Accès aux valeurs d'un tableau

L'accès à une valeur se fait par son indice, les accolades sont obligatoires:

${tableau[2]}

${tableau_asso[deux]}

Opérations sur les tableaux

Voila quelques opérations utiles :

1. **le nombre d'éléments** est retourné par **${#nomtableau[*]}** ou **${#nomtableau[@]}**
2. **tous les éléments** sont retournés comme un seul mot par **${nom-tableau[*]}**
3. **tous les éléments** sont retournés comme des mots séparés par **${nom-tableau[@]}**
4. **tous les indices** sont retournés par **${!nom-tableau[*]}** ou **${!nom-tableau[@]}**
5. l'**ajout d'élement** peut se faire par : **tableau[${#tableau[*]}]=element**
6. La **concaténation de tableaux** peut se faire par cette méthode :
   tableau3=("${tableau1[@]}" "${tableau2[@]}")

4. Interaction avec l'utilisateur

La commande d'écriture : echo

L'affichage de message se fait avec la commande echo :

echo bonjour

echo donner une valeur

La commande de lecture : read

La lecture de valeur au clavier pour les stocker dans une variable se fait avec read

read variable

read permet également d'afficher un message à la lecture :

read -p "Votre Nom" nom

echo Hello $nom

5. Expression arithmétique

L'évaluation d'une expression arithmétique peut se faire avec **deux syntaxes** ou une commande comme **let** ou **expr**:

1. syntaxe à parenthèses : $((expression))
2. syntaxe à crochets $[expression]
3. let est équivalent de ((expression))

Exemple : Exemples d'évaluation d'expressions arithmétiques

```
$ echo $((3*5))
$ 15
$ x=3
$ y=5
$ echo $[ x*y ]
$ 15
$ let n=3*5
$ echo $n
$15

```

6. Structures Conditionnelles

Formes conditionnelles

Les trois formes de la condition if existent sur bash : la **condition simple if** , la **condition alternative if** et la **condition imbriquée if** . La structure conditionnelle imbriquée **case** existe également sur bash

la condition simple if

```
if condition ; then
    instructions
fi
```

Exemple : exemple avec la condition simple

```
#!/bin/bash
read -p "saisir le mot l3in : " mavar
if [ $mavar = "l3in" ] ; then
    echo "correct"
fi
```

Attention : les espaces sont obligatoires autour de l'opérateur =. Les espaces sont obligatoires vers l'intérieur des crochets.

La condition alternative

```
if condition ; then
    Instructions
else
    Instructions Alternatives
fi

```

Exemple : exemple avec la condition alternative

```
#!/bin/bash
read -p "saisir le mot l3in : " mavar
if [ $mavar = "l3in" ] ; then
    echo "correct"
else
    echo "incorrect"
fi
```

La condition imbriquée

```
if condition ; then
    Instructions
elif condition ; then
  Instructions
elif condition; then
...
else
    Instructions Alternatives
fi

```

Exemple : exemple avec la condition imbriquée

```
#!/bin/bash
read -p "saisir le mot l3in ou m1in : " mavar
if [ $mavar = "l3in" ] ; then
    echo "Licence 3 info"
elif [ $mavar = "m1in" ];then
    echo "Master 1 info"
else
    echo "formation inconnue"
fi
```

Test conditionnel

La **condition** sur bash peut être réalisée d'une des **deux manières** suivantes :

1. utiliser la **commande test**, nous vous renvoyer à la page de manuel de cette commande
2. **[expression]**

Vous pouvez aussi utiliser la version étendue de test : [[expression]].

Les opérateurs pouvant être utilisés sont listés dans le tableau suivant :

Opérateurs numériques		**Opérateurs sur les chaînes**		**Opérateurs sur les fichiers**	
Opérateur	Sémantique	Opérateur	Sémantique	Opérateur	Sémantique
-eq	égalité	=	chaînes ident.	-e	existe
-ne	inégalité	!=	chaînes diff.	-f	fichier simple
-lt	inférieur strict.	-z	chaîne vide	-d	répertoire
-le	inférieur ou égal	-n	chaîne non vide	-L	lien symbolique
-gt	supérieur			-s	fichier non

Opérateurs numériques		Opérateurs sur les chaînes		Opérateurs sur les fichiers	
	strict.				vide
-ge	supérieur ou égal			-r	lisible
				-w	modifiable
				-x	exécutable
				-nt	plus récent que
				-ot	plus vieux que
				-O	on est owner du fichier
				-G	on a le même groupe que le fichier

Les opérateurs sur les fichiers présentés ici sont des opérateurs unaires à l'exception de -nt et -ot qui sont des opérateurs binaires. Les opérateurs logiques AND et OR sont respectivement les opérateurs -a et -o pour la commande test (et [expression]). Avec la version étendue [[expression]], ce sont respectivement les opérateurs && et ||.

Exemple : Exemple avec les tests

```
#!/bin/bash
read -p "donner x : " x
if [ $x -ge 10 -a $x -le 20 ];then
    echo "Moyenne"
fi
```

```
#!/bin/bash
read -p "donner x : " x
if test $x -ge 10 -a $x -le 20 ;then
    echo "Moyenne"
fi
```

```
#!/bin/bash
read -p "donner x : " x
if [[ $x -ge 10 && $x -le 20 ]];then
    echo "Moyenne"
fi
```

La condition imbriquée case

```
case EXPRESSION in
    CASE1) Commandes;;
    CASE2) Commandes;;
    ...
    CASEn) Commandes;;
esac

```

Exemple : Exemple avec case

```
1  echo -n "Votre réponse :"
2  read REPONSE
3  case $REPONSE in
4      O* | o*) REPONSE="OUI" ;;
5      N* | n*) REPONSE="NON" ;;
6      *) REPONSE="PEUT-ETRE !";;
7  esac
```

7. Itérations

syntaxe de while

```
1  while CONDITION ; do
2      COMMANDES
3  done
4
```

Exemple : Exemple avec while

```
1  #!/bin/bash
2  while [ $reponse != 'oui' ]; do
3      read -p 'Dites oui : ' reponse
4  done
```

syntaxe de until

```
1  until CONDITION ;do
2      COMMANDES
3  done
4
5
```

Exemple : Exemple avec until

```
1  #!/bin/bash
2  until [ $reponse == 'oui' ];do
3      read -p 'Dites oui : ' reponse
4  done
```

for-do

bash met à disposition **deux syntaxes** de for :

1. la **syntaxe classique** des langages de programmation ;
2. **la syntaxe** de boucle sur une **liste** (**style foreach**).

La syntaxe for classique :

```
1  for ((initial;condition;action)) ; do
2      instructions
3  done
4
```

Exemple : Exemple avec la syntaxe classique de for

```
#!/bin/bash
for ((i=0;i<100;i=i+1)); do
    echo $i
done
```

La syntaxe for sur une liste

```
for Variable in LIST
do
    COMMANDES
done

```

Exemple : Exemples sur des listes

La syntaxe sur liste est très pratique pour faire des traitement sur le résultat de commandes de listing comme ls, find.

```
#!/bin/bash
for i in ‘seq 1 10‘;
do
    echo $i
done
```

```
#!/bin/bash
for variable in 'valeur1' 'valeur2' 'valeur3'
do
    echo "La variable vaut $variable"
done
```

```
for num in {0..9} ; do touch fichier$num.tar.gz ; done
```

```
#!/bin/bash
# script6.sh boucle
#x prend chacune des valeurs possibles correspondant au motif : *.tar.gz
for x in ./*.tar.gz ; do
    # tous les fichiers $x sont renommés $x.old
    echo "$x -> $x.old"
    mv "$x" "$x.old"
done
```

break et continue

La commande **break** permet de quitter la **boucle la plus interne**. La commande **break n** (n entier supérieur à 0) permet de sortir de n boucles imbriquées. La commande **continue** permet d'interrompre l'itération courante d'une boucle et de

passer à l'itération suivante.

8. Fonctions

Définition de fonctions

La définition de fonction peut se faire avec une des deux syntaxes suivantes. La déclaration de fonction connue en C n'existe pas en bash. Une fonction doit être connue avant son appel. Il est donc d'usage de mettre en début de fichier les définitions de fonctions.

```
function nomFonction
{
    instructions ....
}
```

```
nomFonction
{
    instructions ....
}
```

Appel de fonction

Un appel de fonction se fait en donnant juste le nom de la fonction.

```
nomFonction
```

Exemple : Fonction helloworld

```
#!/bin/bash
#helloworld2.sh
#définition de la fonction
function hello
{
    echo "Bonjour tout le monde"
}
#appel de la fonction
hello
```

paramètres de scripts/paramètres de fonction

Dans un script shell ou une fonction, les paramètres suivants sont accessibles :

- $0 : nom du script/de la fonction ;
- $1, $2, ..., $9 : respectivement premier, deuxième, ..., neuvième argument de la ligne de commande (ou d'une fonction) ;
- $# : nombre d'arguments sur la ligne de commande (ou d'une fonction) ;
- $* : tous les arguments vus comme un seul mot
- $@ : tous les arguments vus comme des mots séparés : “$@” équivaut à “$1” “$2” ...

Conclusion

Les shell sont très riches et permettent de faire des tâches et des commandes complexes sur le système à travers leur langage de script. Ils sont particulièrement utiles pour les administrateurs systèmes et réseaux. Reportez vous aux pages de manuel des différents shell pour approfondir vos connaissances sur les scripts.