INFORMATIQUE 1A
Chapitre 2
Programmation Script-Bash

LICENCE 1 - L1MPC

Année académique 2023/2024

USTM - FRANCEVILLE - GABON

Volume horaire : 30 H

BOUITYVOUBOU Henri Mesmin Noël
henri.bouityvoubou@univ-masuku.org

Table des matières

Introduction

La programmation de scripts Bash se réfère à l'écriture de scripts utilisant le langage de script Bash (Bourne Again SHell). Bash est un interpréteur de commandes souvent utilisé dans les systèmes d'exploitation de type Unix, tels que Linux. Les scripts Bash sont des séquences de commandes qui sont exécutées dans un ordre défini pour automatiser des tâches, manipuler des fichiers, traiter des données, ou effectuer d'autres opérations système. Voici quelques caractéristiques importantes de la programmation de scripts Bash :

- **Interprété :** Les scripts Bash sont exécutés par l'interpréteur Bash, ce qui signifie qu'ils ne nécessitent pas de compilation préalable. Cela facilite le développement et la modification rapide des scripts.
- **Gestion des Processus :** Il gère l'exécution des programmes et des processus sur l'ordinateur, en coordonnant l'utilisation du processeur et de la mémoire.
- **Shell scripting :** Bash est un shell, ce qui signifie qu'il sert d'interface entre l'utilisateur et le système d'exploitation. Les scripts Bash permettent aux utilisateurs de combiner des commandes shell pour effectuer des tâches complexes.
- **Contrôle de flux :** Les scripts Bash incluent des structures de contrôle de flux telles que les boucles (for, while) et les structures conditionnelles (if, else) pour permettre la prise de décisions et l'itération.
- **Variables :** Les scripts Bash utilisent des variables pour stocker des valeurs. Les variables peuvent être définies et manipulées pour effectuer des opérations sur les données.
- **Fonctions:** Les scripts Bash peuvent inclure des fonctions pour organiser le code en blocs réutilisables.
- **Expressions régulières :** Bash prend en charge l'utilisation d'expressions régulières pour effectuer des opérations de recherche et de remplacement dans les chaînes de texte.

Un script Bash typique commence par une ligne spéciale, appelée shebang, qui indique le chemin vers l'interpréteur Bash. Ensuite, le script consiste en une séquence de commandes Bash.

```
# !/bin/bash
echo "Bonjour, quel est votre nom ?"
read nom

echo "Quel est votre AGE ?"
read age

echo "$nom vous avez $age ans"

# Exemple de boucle
for i in {1..5}
do
  echo " $i"
done
```

Dans le cadre de ce cours, nous allons essentiellement étudier la programmation script en bash.

Interpréteurs de commandes

Les interpréteurs de commandes permettent aux utilisateurs d'interagir avec le système d'exploitation en entrant des commandes textuelles. Ils sont utilisés pour lancer des programmes, naviguer dans le système de fichiers, gérer les processus, et effectuer diverses tâches système. Les scripts Bash, par exemple, sont des séquences de commandes Bash interprétées par l'interpréteur Bash.
Il en existe différents interpréteurs de commandes:

1. Bash (Bourne Again SHell) : Bash est l'un des interpréteurs de commandes les plus populaires et est largement utilisé dans les systèmes basés sur Unix, y compris Linux. Il est compatible avec le shell Bourne original et offre de nombreuses fonctionnalités supplémentaires, telles que des fonctions, des boucles, et des expressions régulières.

2. Sh (Bourne Shell) : C'est le shell original développé par Stephen Bourne. Bien que moins puissant que Bash, il reste présent sur de nombreux systèmes Unix.

3. Zsh (Z Shell) : Zsh est un shell avancé avec de nombreuses fonctionnalités supplémentaires par rapport à Bash, comme la correction automatique des erreurs de frappe, des thèmes personnalisables, et une meilleure gestion des fichiers.

4. Fish (Friendly Interactive SHell) : Fish se distingue par son approche conviviale pour les utilisateurs. Il propose une auto-complétion interactive, une coloration syntaxique améliorée, et d'autres fonctionnalités conviviales.

5. Dash : Dash est un shell minimaliste conçu pour être rapide et léger. Il est souvent utilisé comme /bin/sh sur certains systèmes Unix.

6. PowerShell : Bien que principalement associé à Microsoft Windows, PowerShell est un shell et un langage de script puissant qui est également disponible pour certaines distributions Linux.

7. tcsh (Tenex C Shell) : tcsh est une version améliorée du C Shell (csh) avec des fonctionnalités supplémentaires telles que l'historique des commandes et la complétion automatique.

Un interpréteur de commandes (shell en anglais) est un programme qui agit en tant qu'interface utilisateur pour l'accès au système d'exploitation. Il permet aux utilisateurs d'entrer des commandes textuelles qui sont interprétées et exécutées par le système d'exploitation. Les interpréteurs de commandes sont une composante essentielle des systèmes d'exploitation de type Unix, tels que Linux, macOS, et d'autres variantes d'Unix.

Principales commandes utiles

cat : concatenate files and print on the standard output

- cat [OPTION]... [FILE]...

Les options peuvent prendre différentes valeures telles que :

- -n, –number : number all output lines
- -E, –show-ends : display $ at end of each line

cut : remove sections from each line of files

- cut [OPTION]... [FILE]...

Les options peuvent prendre différentes valeures telles que :

- -d, –delimiter=DELIM : use DELIM instead of TAB for field delimiter
- -f, –fields=LIST : select only these fields; also print any line that contains no delimiter character, unless the -s option is specified

head : output the first part of files

- head [OPTION]... [FILE]...

Les options peuvent prendre différentes valeures telles que :

- -c, –bytes=[-]NUM : print the first NUM bytes of each file;
- -n, –lines=[-]NUM : print the first NUM lines instead of the first 10;

tail : output the last part of files

- tail [OPTION]... [FILE]...

Les options peuvent prendre différentes valeures telles que :

- -c, –bytes=[+]NUM : output the last NUM bytes;
- -n, –lines=[+]NUM : output the last NUM lines, instead of the last 10;

echo : display a line of text

- echo [SHORT-OPTION]... [STRING]...

Les options peuvent prendre différentes valeures telles que :

- -n : do not output the trailing newline
- -e : enable interpretation of backslash escapes
- -E : disable interpretation of backslash escapes (default)

Shell-Script (bourne shell)

Les scripts Bash sont largement utilisés pour l'automatisation des tâches système, la gestion de fichiers, la configuration du système, et d'autres opérations liées à l'administration système.

- Shell :
  1. c'ets l'interface de l'utilisateur pour lancer des commandes
  2. il permet d'executer des scripts
- Shell-script :
  1. c'est un fichier de commandes
  2. qui sera appelé comme une commande de base : monscript.sh
  3. ce fichier porte l'extension .sh
  4. qui contient en début de lignes, la spécification de de l'interpréteur utilisé
  5. le fichier doit disposer des droits d'exécution (chmod 777 monscript.sh)

Cette section décrit par ordre de préférence les éléments de base utiles pour cette formation.

Variables

les variables sh

- $HOME : répertoire de base
- $PATH : repertoires de re her hes des
- $PS1 , $PS2 : prompts
- $# : nombre de paramètres du script
- $0 à $9 : parametres 0 (nom de la ommande) à 9
- $* : tous les parametres
- $? : code de retour (erreur) de la dernière commande

```
#! /bin/sh
echo "commande $0"
echo "avec  $#  arguments"
echo "avec  $$ processus"
echo "avec pour parametre 1 = $1"
echo "avec pour arguments : $*"
```

Accès aux variables

L'accès aux variables se fait directement par appel précédé du caractère $ suivi du nom de la variable. Exemple :

```
#! /bin/sh
var = 150
nom = "Henri"
echo "$var"
echo "$nom"
```

Entrées/Sorties

Dans cet exemple, je vous montre comment lire au clavier une valeur et afficher le résultat à l'écran :

```
#! /bin/sh
echo "entrez un nombre : "
read var
echo "vous avez saisi : $var"
```

L'entrée standard ici est considérée comme le clavier et la sortie comme étant l'écran.

Fonctions

```
#! /bin/sh
arguments(){
    echo " fonction arguments : $*"
}
echo "dans le shell $*"
arguments "hello my friend"
```

```
monscript.sh bonjour
bonjour
hello my friend
```

Evaluations

expr

Pour calculer la valeur d'une expression, on peut utliser aussi la commande expr de la façon suivante :

1. Expressions booléennes :

```
        expr 45 != 20
        expr 45 = 20
        expr 45 < 20
        expr 45 > 20
```

2. Expressions arithmétiques :

```
        expr 45 - 20
        expr 2 + 18
        expr 45 * 20
        expr 45 / 9
```

3. Modification de la variable a :

```
        a = 10
        a=‘expr $a + 1‘
        a=‘expr $a - 1‘
        a=‘expr $a * 2‘
```

eval

Exécution par le shell en deux phases :

- Evaluation des arguments de eval
- Réévaluation des arguments et exécuution de la commande

```
% var= '$val'
% echo $var
$
% val='Unix'
% eval echo $var
```

Exemple : accès au dernier argument d'un script (inférier à 9)

```
eval dernier=\$"$# "
echo $dernier
```

test

```
#! /bin/sh
var=10
while test $var -ge 4
do
e ho " $var"
var=‘expr $var - 1‘
done
```

Structures de contôle

boucle Pour : for ... do ... done

```
#! /bin/sh
for i in Lenine Staline Krout hev Brejnev
do
    echo " $i fut maitre de la Russie !"
done
echo " -"
# aussi possible : for i in *
for i in ‘ls *‘
do
    echo " $i est un fichier"
done
```

```
#! /bin/sh
for i in $*
do
    echo $i
done
```

```
#! /bin/sh
for i in $@
do
echo $i
done
```

- $@ == $* : les arguments de la sommande sont réexaminées
- "$*" est un seul mot composé de tous les arguments
- "$@" : fournit exactement les arguments

boucle Tant que : while ... do ... done

```
#! /bin/sh
var=10
while test $var -ge 4
do
    echo " $var"
    var=‘expr $var - 1‘
done
```

Condition : if ... then ..... else .... fi

```
#! /bin/sh
if test $#  -ne 1
then
e ho " erreur nombre d'arguments" 1>&2
exit 1
fi
```

```
#! /bin/sh
FILE=$1
if test ! -f $FILE
then
    echo " erreur : $FILE n'existe pas" 1>&2
    exit 1
else
    echo " $FILE existe, je l'ai vu "
fi
```

Branchement : case ... esac

```
#! /bin/sh
case $file in
*.p)
    echo "$file is a pas al program" ;;
*. )
    echo "$file is a C program" ;;
*. |*. ++)
```

```
    echo "$file is a C++ program" ;;
*)
    echo "I'm rather puzzled"
    echo "
    with that $file " ;;
esac
```

lecture de fichiers

Traiter séquentiellement une à une, les lignes d'un fichier

```
#! /bin/sh
while read laligne
do
# traitement de la ligne
...
done < $file
```

```
#! /bin/sh
nlignes=`w -l < $file`
i=1
while test $i -le $nlignes
do
    laligne=`readline $file $i`
    # traitement de la ligne
    ...
    i=`expr $i + 1`
done
```