Module Linux

Equipe Pédagogique

S. BEN YAALA
H. SLIMANI

Programmation SHELL

2019-2020

Plan

- Définition d’un shell
- Types de shell
- Personnaliser l’environnement bash
- Variables d’envrionnement
- Scripts shell
- Opérateurs
- Structures itératives et conditionelles

Définition d'un Shell

Un shell assure deux rôles principaux :

1. C'est un interpréteur de commandes
2. C'est un langage de programmation :

Le Shell comme étant interpréteur des commandes :

- Ses fonctionnalités sont :

- Affichage de l'invite de commande ou prompt ($) d'attente de lecture au clavier.

- Lecture d'une commande (validée par RETURN ou ENTRÉE).

- Analyse syntaxique (découpage en mot).

- Interprétation des caractères spéciaux.

- Exécution de la commande et retour au début.

Le Shell comme étant un langage de programmation

- C’est un langage de programmation script.
- Il intègre les notions de :

  - variables,

  - opérateurs

  -Structures de contrôle

Shells :quelques exemples

- Bourne shell (/bin/sh): shell standard unix, de AT&T Bell Laboratories
- Korn shell (/bin/ksh): de David G. Korn dérivé du Bourne shell
- C shell(/bin/csh) de Berkeley BSD
- bash (Bourne again shell) : GNU :le Shell par défaut sous Linux. Il est conforme à la norme IEEE POSIX P1003.2/ISO 9945.2

Différences entre /bin/bash et /bin/sh


```
[root@localhost ~]# ls -l /bin/bash
-rwxr-xr-x. 1 root root 960392  2 août   2016 /bin/bash
[root@localhost ~]# ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 16 févr. 12:28 /bin/sh -> bash
[root@localhost ~]#
```

```
sahar@ubuntu:~$ ls -l /bin/bash
-rwxr-xr-x 1 root root 1021112 May 16  2017 /bin/bash
sahar@ubuntu:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Sep 18 08:18 /bin/sh -> dash
sahar@ubuntu:~$
```

Personnaliser l’environnement BASH

Il y’a deux types de fichiers de configuration utilisés pour la personnalisation de l'environnement :

- Les fichiers qui sont lus au moment de la connexion (login)
- Les fichiers qui sont lus à chaque lancement d’un shell

Fichiers lus au moment de la connexion

- **/etc/profile** : commun à tous les utilisateurs (contient le umask)
- **~/.bash_profile , ~/.bash_login** ou **~/.profile** spécifiques à chaque utilisateur
- **~/. bash_history** : L'historique des commandes tapées

Fichiers lus à chaque lancement de shell

- **/etc/bashrc ou /etc/bash.bashrc** : commun à tous les utilisateurs
- **~/.bashrc** spécifique à chaque utilisateur (on peut trouver et ajouter les alias ici).

Variable d'environnement

- Une variable d'environnement est une variable accessible par tous les processus fils du shell courant. Pour créer une variable d'environnement, on exporte la valeur d'une variable avec la commande export.

```
export   variable
```

- Pour illustrer la différence entre une variable locale et une variable d'environnement, il suffit de créer une variable d'environnement, de lancer un shell fils et d'afficher la variable.

```
$ ma_variable=toto
$ export ma_variable
$ sh
$echo $ma_variable
toto
 $ exit
```

- Pour supprimer cette variable: **unset ma_variable**

Variable d’environnement

- Pour afficher toutes les variables d’environnement, il faut utiliser la commande **env**
- Pour afficher le contenu, il faut lancer **echo $NOM_VAR_MAJ**:

HOME : Répertoire Home de l'utilisateur courant.

PATH : Command search path

PWD : répertoire courant (OLDPWD : ancien répertoire)

LOGNAME : nom de connexion

PS1 : invite primaire du shell ($)

PS2 : invite secondaire du shell pour les commandes incomplètes

PS3 : invite de l'instruction select (#?)

TERM : type du terminal

MAIL : nom du fichier de messagerie (ex: /var/mail/$LOGNAME)

Débuter avec les scripts shells

- Un script shell est un simple fichier texte exécutable (avec un droit d‟exécution x)
- Il doit impérativement commencer par une ligne indiquant au système le shell qu'il faut utiliser :

Shebang → **#!** **/chemin/interpréteur**

Exemple : /bin/bash ou /bin/sh

- Le script doit être rendu exécutable : **chmod +x fichier**
- Dans un script shell on peut avoir ; des variables, des structures de contrôles, des structures répétitives...etc d'où l'appellation ***script shell***

Exécution du script

- Il est possible d'exécuter les scripts avec les méthodes suivantes :

- Méthode 1: $ shell script

Exécution par le shell cité (script fichier texte non nécessairement exécutable : droits x positionnés ou non)

- Méthode 2 :

- rendre le script exécutable exemple **chmod +x script.sh**

- exécution : **./script.sh**

Exemple : Premier script schell

- Ecrire un script qui affiche bonjour tekup

Solution

❑ Edition du script

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
  GNU nano 2.3.1                Fichier : script1.sh

#!/bin/bash
echo "bonjour tekup"
```

❑Exécution

- Méthode 1

```
[root@localhost ~]# ls -l script1.sh
-rw-r--r--. 1 root root 33  1 mars  08:56 script1.sh
[root@localhost ~]# bash script1.sh
bonjour tekup
[root@localhost ~]#
```

- Méthode 2

```
[root@localhost ~]# chmod +x script1.sh
[root@localhost ~]# ./script1.sh
bonjour tekup
[root@localhost ~]#
```

Les Entrées-Sorties

- Ce sont les voies de communication entre le programme bash et la console :

- **echo**: affiche l'argument texte entre guillemets sur la sortie standard

- **read:**

  - permet l'affectation directe par lecture de la valeur, saisie sur l'entrée standard au clavier.

  - Lorsque la commande **read** est utilisée sans argument, la ligne lue est enregistrée dans la variable prédéfinie du shell **REPLY** .

  Exemple : read var1 var2 ... attend la saisie au clavier d'une liste de valeurs pour les affecter, après la validation globale, respectivement aux variables var1, var2 ..

Exemple 1

Ecrire un script appelé script1 qui demande votre nom et prénom et affiche

bonjour votre_nom votre_prenom.

Solution


```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
  GNU nano 2.3.1                Fichier : fichier

#!/bin/bash
echo "Entrez votre nom : "
read nom
echo "Entrez votre prenom : "
read prenom
echo "Bonjour  $nom $prenom "
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[root@localhost ~]# nano fichier
[root@localhost ~]# bash fichier
Entrez votre nom :
tekup
Entrez votre prenom :
student
Bonjour  tekup student
[root@localhost ~]#
```

Variables prédéfinies spéciales pour le passage de paramètres :

- Ces variables sont automatiquement affectées lors d'un appel de script suivi d'une liste de paramètres.

Variable	**Interprétation**
$?	C'est la valeur de sortie de la dernière commande. Elle vaut 0 si la commande s'est déroulée sans problème.
$0	Cette variable contient le nom du script
$1 à $9	Les (éventuels) premiers arguments passés à l'appel du script
$#	Le nombre d'arguments passés au script
$*	La liste des arguments à partir de $1
$$	le n° PID du processus courant
$!	le n° PID du processus fils

Exemple 2

- Ecrire un script shell pour lequel vous passez au moment d'exécution les paramètres suivants : tic-b, tic-c,tic-d,tic-e,tic-f,tic-g. tic-h,tic-i

Ensuite afficher :

- Le premier argument
- Le dernier argument,
- Le nom du script
- La Liste de tous les arguments
- Le nombre des arguments

Solution


```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
  GNU nano 2.3.1                Fichier : script3.sh

#!/bin/bash
echo "le premier argument est :$1"
echo "le dernier argument est : $8"
echo "le nom du script est:$0"
echo "La Liste de tous les arguments :$*"
echo "le nombre des arguments est :$#"
```

```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
[root@localhost ~]# chmod +x script3.sh
[root@localhost ~]# ./script3.sh tic-b tic-c tic-d tic-e tic-f tic-g tic-h tic-i
le premier argument est :tic-b
le dernier argument est : tic-i
le nom du script est:./script3.sh
La Liste de tous les arguments :tic-b tic-c tic-d tic-e tic-f tic-g tic-h tic-i
le nombre des arguments est :8
[root@localhost ~]#
```

Exemple 3 :

Ecrire un script shell qui permet d'afficher la liste des processus actifs et la date actuelle sous cette forme

La liste des processus actifs est :

La date est :

Solution


```
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
  GNU nano 2.3.1                Fichier : script.sh

#!/bin/bash
echo "la liste des processus actifs est `ps`"
echo "la date est`date`"
```

```
[root@localhost ~]# bash script.sh
la liste des processus actifs est    PID TTY          TIME CMD
  4595 pts/0    00:00:01 su
  4601 pts/0    00:00:00 bash
  4786 pts/0    00:00:00 bash
  4787 pts/0    00:00:00 ps
la date estdim. mars  1 10:30:57 PST 2020
[root@localhost ~]#
```

Les opérateurs arithmétiques

Opérateur	Rôle
+	Addition
-	Soustraction
*	Multiplication
**	exponentiation : a^b =a**b
/	Division
%	Modulo

Exemple :

```
[sahar@localhost ~]$ a=9
[sahar@localhost ~]$ b=10
[sahar@localhost ~]$ echo $((a+5))
14
[sahar@localhost ~]$ echo $((a-5))
4
[sahar@localhost ~]$ echo $((a**2))
81
```

```
[sahar@localhost ~]$ A=5
[sahar@localhost ~]$ B=6
[sahar@localhost ~]$ ((somme=A+B))
[sahar@localhost ~]$ echo $somme
11
[sahar@localhost ~]$ ((produit=A*B))
[sahar@localhost ~]$ echo $produit
30
```

Exemple

- Ecrire un script arith.sh qui lit à partir du clavier deux nombres et un opérateur arithmétique . Ensuite il fait le calcul.
- Exemple :

donner a 5

donner b 6

donner l'operateur +

5+6 =11

Solution


```
  GNU nano 2.3.1                Fichier : arith.sh

#!/bin/bash
echo "donner la valeur de a"
read a
echo "donner la valeur de b"
read b
echo "donner l'opérateur"
read operateur

echo "$a $operateur $b"=$(($a $operateur $b))
```

```
[root@localhost ~]# bash arith.sh
donner la valeur de a
5
donner la valeur de b
6
donner l'opérateur
+
5 + 6=11
[root@localhost ~]#
```

La commande expr


```
[sahar@localhost ~]$ a=3
[sahar@localhost ~]$ expr $a + 5
8
[sahar@localhost ~]$ b=`expr $a - 5`
[sahar@localhost ~]$ echo $b
-2
[sahar@localhost ~]$
```

Les opérateurs logiques

Opérateur	Rôle
a&&b	ET logique
a||b	OU logique
!a	négation logique

Les opérateurs de comparaison

Opérateur	Syntaxe
!=	**if ["$a" != "$b"]**
=	est égal à if ["$a" = "$b"] pour les chaines Si non, pour les entiers if ["$a" == "$b"]
<	**if [["$a" < "$b"]]** **if ["$a" \< "$b"]**
>	**if [["$a" > "$b"]]** **if ["$a" \> "$b"]**

Comparaison sans l'instruction test

Opérateur	Rôle
-eq	est égal à **if ["$a" -eq "$b"]**
-ne	n'est pas égal à **if ["$a" -ne "$b"]**
-gt	est supérieur à **if ["$a" -gt "$b"]**
-ge	est supérieur ou égal à **if ["$a" -ge "$b"]**
-lt	est inférieur à **if ["$a" -lt "$b"]**
-le	est inférieur ou égal à **if ["$a" -le "$b"]**

Structures de contrôle

+

La structure de contrôle : if

- Syntaxe

```
if   cmd1
then     Instructions1;
elif  cmd2
then    Instructions2;
else     Instructions4;
fi
```

- Interprétation

Si la commande **cmd1** se passe bien *(retourne la valeur 0)* alors les instructions1 qui se trouvent après le mot clé **then** sont exécutées, sinon, si la commande **cmd2** se passe bien *(retourne la valeur 0)* alors ce sont les instructions2 qui sont exécutées; Sinon exécutions des instructions4.

L’instruction conditionnelle : test

- Elle constitue l’indispensable complément de l’instruction if.
- Elle permet :
  – de reconnaître les caractéristiques des fichiers et des répertoires,
  – de comparer des chaînes de caractères,
  – de comparer algébriquement des nombres.

Prédicats sur les fichiers

Opérateur	Rôle
test -r fichier	vrai si fichier existe et on a le droit read
test -w fichier	vrai si fichier existe et on a le droit write
test -x fichier	vrai si fichier existe et on a le droit eXecute
test -e fichier	vrai si le fichier existe
test -f fichier	vrai si fichier existe et c'est un fichier ordinaire
test -d fichier	vrai si fichier existe et c'est un répertoire

Prédicats sur les chaines de caractères

on note ch1 et ch2 deux chaînes de caractères.

Opérateur	Rôle
test ch1 = ch2	vrai si ch1 est égale à ch2.
test ch1 != ch2	vrai si ch1 est différente de ch2.
test -n ch1	vrai si ch1 est non vide
test -z ch1	vrai si ch1 est vide

Prédicat sur les entiers

Opérateur	Rôle
test n1 -eq n2	vrai si n1 est égale à n2 (equal to)
test n1 -ne n2	vrai si n1 est différent de n2 (not equal to)
test n1 -gt n2	vrai si n1 est strictement supérieur à n2 (is greather than)
test n1 -ge n2	vrai si n1 est supérieur ou égale à n2 (is greater than or equal to)
test n1 -lt n2	vrai si n1 est strictement inférieur à n2 (is less than)
test n1 -le n2	vrai si n1 est inférieur ou égale à n2 (is less than or equal to)

La boucle for

	Forme 1	Forme 2	Forme 3
Syntaxe	**for** variable **in** ch1 ch2... chn **do** Commandes **done**	**for** variable **do** Commandes **done**	**for** variable **in** * **do** commandes **done**
Interprétation	les valeurs de variable sont les chaines de ch 1 à ch n	variable prend ses valeurs dans la liste des paramètres du script.	la liste des fichiers du répertoire constitue les valeurs prises par variable.

Forme 1


```
  GNU nano 2.3.1                Fichier : script.sh

#!/bin/sh
for a in lundi mardi mercredi
do
echo $a
done
```

```
[sahar@localhost ~]$ sh script.sh
lundi
mardi
mercredi
[sahar@localhost ~]$
```

Forme 2


```
 GNU nano 2.3.1                Fichier : script.sh

#!/bin/sh
for a
do
echo $a
done
```

```
[sahar@localhost ~]$ chmod +x script.sh
[sahar@localhost ~]$ ./script.sh tekup university
tekup
university
[sahar@localhost ~]$ sh script.sh lundi mardi merecredi
lundi
mardi
merecredi
[sahar@localhost ~]$ sh script.sh lundi 1 2 merecredi
lundi
1
2
merecredi
```

Forme 3


```
 GNU nano 2.3.1                Fichier : script.sh

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

```
[sahar@localhost ~]$ sh script.sh
Bureau
Documents
Images
Modèles
Musique
Public
script.sh
Téléchargements
Vidéos
[sahar@localhost ~]$
```

Itération while

- Elle correspond à l'itération *tant que*.
- Syntaxe

```
while suite_cmd1
do
   suite_cmd2
done
```