MAI – L2/S4

Systèmes d'exploitation

Séquence 4 : Programmation shell

Cheikh Sidy Mouhamed CISSE Enseignant Associé en Informatique à l'UVS

Chapitre 4 : Programmation shell

Objectifs spécifiques : A la suite de cette séquence, l'étudiant doit être capable de:

1. connaître l'utilisation des commandes Shell sous linux.
2. Savoir programmer des Scripte sous Linux

Présentation générale

- Le shell est:
  - Un interpréteur de commandes
  - Langage de programmation

- Sous Linux, un fichier contenant des commandes est appelé script

Présentation générale

- Le shell comporte des instructions et des variables
- Les noms de variables sont des chaînes de caractères;
- Leurs contenus sont également des chaînes de caractères.
- Si un script commence par la ligne #!/bin/xxx, il est interprété par le shell /bin/xxx.
- **Exemple:**

#!/bin/bash
/bin/sh si vous souhaitez coder pour sh
/bin/ksh pour ksh
etc

Déclaration de variables

- L'assignation d'une valeur à une variable se fait par un nom;
- La référence à cette valeur se fait par son nom précédé du caractère $, comme dans:

```
mavariable=bonjour    // assignation
echo $mavariable      // référence
```

- Les mécanismes de tubes et de redirections sont utilisables dans un script.

Déclaration de variables

- Affecter une valeur à une variable

```
cisse@cisse-laptop:~$ var1="cisse cheikh"
cisse@cisse-laptop:~$ echo $var1
cisse cheikh
```

- **Exemples**
- Les commandes *uname* et *logname* sont remplacées par leur résultat avant exécution de la commande *echo*

```
cisse@cisse-laptop:~$ echo vous êtes actuellement connecté sur la machine $(uname -n) et vous etes $(logname)
vous êtes actuellement connecté sur la machine cisse-laptop et vous etes croot
cisse@cisse-laptop:~$
```

Installer un nouveau shell

- Par défaut, *sh* et *bash* sont installés sur les systèmes linux.

- Pour installer un autre shell, comme *ksh* par exemple:

 # apt-get install ksh

- Pour l'utiliser à un compte, il faut taper la commande:

$ chsh

- Puis indiquer où se trouve le programme qui gère le shell

```
cisse@cisse-laptop:~$ chsh
Password:
Changing the login shell for cisse
Enter the new value, or press ENTER for the default
        Login Shell [/bin/bash]: /bin/ksh
```

- */bin/ksh* pour ksh, */bin/sh* pour sh, */bin/bash* pour bash, etc

La programmation de base en shell

- Création avec l'éditeur *nano* ou *vim* ou n'importe quel éditeur de texte du fichier *teste* contenant la ligne *ls –aCF.*

$nano teste.sh

```
#!/bin/bash
ls –aCF
```

$chmod a+x teste //ajoute le droit x

$./teste.sh //exécution

- **NB:**

Par convention on donne l'extension *.sh* pour indiquer que c'est un script shell, mais sachez que ce n'est pas une obligation. Certains scripts shell n'ont d'ailleurs pas d'extension du tout. On pouvait appeler notre script teste tout cour

Le passage des paramètres

- Le script teste ne s'applique qu'au répertoire courant
- On peut lui attribuer des paramètres comme le nom d'un répertoire, par exemple.
- Modifions le fichier teste.sh:

Le passage des paramètres

$nano teste.sh

#!/bin/bash

echo « contenu du repertoire $1»

ls –aCF $1

```
cisse@cisse-laptop:~$ ./teste /tmp
contenu du repertoire /tmp
./            .ICE-unix/        pulse-PKdhtXMmr18n/    .X0-lock
../           keyring-3UAuCr/   pulse-ZTV2h7UgQKlp/    .X11-unix/
.esd-1000/    orbit-cisse/      ssh-DnotkM1180/
.esd-114/     orbit-gdm/        virtual-cisse.GRk6YW/
cisse@cisse-laptop:~$
```

Le passage des paramètres

- Paramètres (arguments) du script
  - Chaînes de caractères séparées par des blancs, apparaissant après le nom du script au moment de son appel.
- Variables prédéfinies

Variable	Fonction
$$	L'identifiant du processus courant (PID)
$?	Le code retour de la commande précédente
$!	L'identifiant du processus fils
$0	Le nom du script
$1 à $9	Les neuf premiers arguments du script
$#	Le nombre d'arguments
$*	Tous les argument (à partir de $1), séparés par des espaces

Le passage des paramètres

- Évaluation d'une expression arithmétique
  - Remplacer une expression par la valeur de son résultat
  - A l'aide de la notation **$((expression)) ou $[expression]**

```
cisse@cisse-laptop:~$ un=1
cisse@cisse-laptop:~$ deux=2
cisse@cisse-laptop:~$ echo $[ $un + $deux]
3
cisse@cisse-laptop:~$
```

Structures conditionnelle

- Les structures de contrôle conditionnelle permettent d'effectuer une action en fonction d'une expression logique.

- **Syntaxe 1**

```
if cmd
then
    instructions;
fi
```

- Si la commande *cmd* se passe bien (retourne la valeur 0) alors les instructions qui se trouvent après le mot clé *then* sont exécutées

Structures conditionnelle

- **Syntaxe 2**

```
if cmd
then
    instructions;
else
    instructions;
fi
```

- Si la commande ***cmd*** se passe bien (retourne la valeur 0) alors les instructions qui se trouvent après le mot clé ***then*** sont exécutées sinon ce sont les instructions qui se trouvent après le mots clé ***else*** qui sont exécutées

Structures conditionnelle

- **Syntaxe 3**

```
if cmd1
then
    instructions1;
elif cmd2
then
    instructions2;
else
    instructions3;
fi
```

- 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 *instructions3.*

Structures conditionnelle

- **Syntaxe 4**

```
if cmd
then
    instructions;
elif cmd
then
    instructions;
elif cmd
then
    instructions;
else
    instructions;
fi
```

- Le fonctionnement reste le même que dans la syntaxe 3. Mais on a ajouté une ligne *elif cmd*. Il est possible d'en rajouter autant que nécessaire. Attention cependant à garder un code lisible et compréhensible.

Structures conditionnelle: OU logique

- **Syntaxe 5**

```
if cmd1 || cmd2
then
    instructions;
fi
```

- Dans cette syntaxe, l'opérateur ***logique OU*** est utilisé. Il est représenté par la séquence de caractères **||**.La condition *if* sera vérifier si l'une des commandes *cmd1* ou *cmd2* retourne sans erreur (code retour 0).

Structures conditionnelle : ET logique

- **Syntaxe 6**

```
if cmd1 && cmd2
then
    instructions;
fi
```

- Dans cette syntaxe, l'opérateur ***logique ET*** est utilisé. Ce dernier est représenté par la séquence de caractères ***&&.*** La condition if sera vérifier si les commandes ***cmd1*** et ***cmd2*** retourne la valeur vrai (code retour 0)

Structures conditionnelle

Exemple

Nous souhaitons comparer le contenu de la variable *$var* avec la chaîne de caractères « *bonjour* » et afficher ***ok*** sur la sortie standard si ces dernières sont identique.

```
if [ $var = "bonjour" ]
    then
        echo "ok";
 fi
```

Quelques opérateurs

Quelques opérateurs					
Comparaison numérique		Comparaison de chaînes		Test de fichiers et répertoires	
Opérateur	Sens	Opérateur	Sens	Opérateur	Sens
-eq	Égal	-z	Chaîne vide	-L	Lien symb.
-ne	Différent	-n	Chaîne non vide	-d	Répertoire
-lt	Inférieur	=	Chaînes ident.	-e	Existe
-le	Inf. ou ég.	!=	Chaînes diff.	-f	Fic. ordinaire
-gt	Supérieur			-s	Fic. non vide
-ge	Sup. ou ég.			-r	Fic. lisible
				-w	Fic. modifiable
				-x	Fic. exécutable
				-nt	Plus récent que
				-ot	Plus vieux que

Structures conditionnelle: case

- **Case**

```
case     expression          in
     case1) liste_de_commandes ;;
     case2) liste_de_commandes ;;
     ...
     casen) liste_de_commandes ;;
esac
```

- **Exemple:**

```
echo -n "votre réponse :"
read reponse
case $reponse in
     o* | o*) reponse="oui" ;;
     n* | n*) reponse="non" ;;
     *) reponse="peut-etre !" ;;
esac
```

Structures conditionnelle: les boucles

- **boucle for**

```
for variable in list
do commandes
done
```

- **Boucles while**

```
while CONDITION
do COMMANDES
done
```

Structures conditionnelle

- **boucle until**

```
until   CONDITION
do      COMMANDES
done
```

Structures conditionnelle

Répétition

- Commande **break**
  - Permet de quitter la boucle la plus interne

- Commande **break n**
  - **n**: entier supérieur à 0
  - Permet de sortir de **n** boucles imbriquées.

- Commande **continue**
  - Permet d'interrompre l'itération courante d'une boucle et de passer à l'itération suivante.

Quelques exemples

- Vérifier si un fichier est accessible en lecture
  - Pour vérifier si un fichier est lisible, il est nécessaire d'utiliser l'option -r.

```
if [ -r /etc/passwd ]
      then
            echo "le fichier /etc/passwd est accessible en lecture"
fi
```

- Vérifier si un fichier est accessible en écriture
  - Pour vérifier si un fichier est écriture (afin de savoir si il est modifiable) vous pouvez utiliser l'option l'option –w

```
if [ -w /etc/passwd ]
      then
            echo "le fichier /etc/passwd est accessible en écriture"
fi
```