Cours UNIX

Chapitre 6

Scripts

Généralités

➔ Qu'est ce qu'un script ?

Les scripts sont des suites de commandes exécutées par un shell. Les commandes internes d'un shell forment ainsi un véritable langage de programmation, souvent appelé *langage de commandes*.

Ces scripts sont surtout utilisés par les administrateurs réseaux et les développeurs car ils permettent de créer très rapidement des programmes, des nouvelles commandes ou des "moulinettes".

Tous les concepts vus dans le chapitre 3, *interpréteurs de commandes*, sont utilisables à l'intérieur d'un script, comme par exemple, les variables locales et d'environnement, les substitutions, le lancement de processus, ...

➔ Syntaxe

Dans un script, un commentaire commence par le caractère # et finit à la fin de la ligne.

Exemples:

```
# ceci est un commentaire
commande1 & # lancement de la commande1
```

En règle générale, un script commence par le chemin du shell avec lequel il doit être exécuté (si ce n'est pas le cas, c'est le shell à partir duquel le script est lancé qui s'en charge) précédé des caractères `#!`.

Exemples:

```
Pour utiliser le Bourne Shell: #!/bin/sh
Pour utiliser le C-Shell: #!/bin/csh
```

➔ Exécution

Il existe plusieurs manières d'exécuter un script:

- le rendre exécutable:

```
$ chmod +x monscript
$ monscript
```

- passer sont nom en paramètre d'un shell:

```
$ sh monscript
```

- utiliser une fonction de lancement de commande du shell:

```
$ . monscript
$ source monscript
```

Arguments

Les arguments sont les paramètres tapés après le nom du script exécuté. Ils sont accessibles et manipulables au travers de variables prédéfinies:

```
$#     nombre d'arguments
$*     liste de tous les arguments
$0     nom du script en cours d'exécution
$1     premier argument
$2     deuxième argument
$3     troisième argument
et ainsi de suite...
```

Exemple:

```
$ cat monscript1
#!/bin/sh
echo Premier argument de la commande $0: $1

$ ./monscript1 tutu
Premier argument de la commande ./monscript1: tutu
```


Structures de contrôle (Bourne Shell)

Les structures de contrôles utilisent la commande externe `test` pour exécuter des expressions booléennes. Celle-ci est décrite dans la dernière partie de ce chapitre.

➔ if

Syntaxe :

```
if expression
then
       commandes
else
       commandes
fi
```

Exemples :

```
if test -z "$machaine"
then
       echo "machaine est vide"
fi

if [ `who | grep '^toto' | wc -l` -gt 0 ]
then
       echo "L'utilisateur toto est connecté"
else
       echo "L'utilisateur toto n'est pas connecté"
fi
```

➔ for

La structure *for* du Bourne Shell ne correspond en rien aux structures *for* que l'on peut rencontrer dans des langages comme le C, le Basic ou Java. Ici le *for* sert à parcourir une *liste de mots* (à la manière d'un foreach que l'on trouve dans les langages PHP et Perl).

Syntaxe:

```
for variable in liste de mots;
do
       commandes
done
```

La liste de mots peut être remplacée par les *caractères d'expansion* du shell (voir chapitre 3), dans ce cas, la liste est constituée des fichiers trouvés dans le répertoire courant.

Exemples:

```
for mavar in toto tutu titi tata;
do
       echo "le mot: $mavar"
done

for fichier in *.c;
do
       echo "Compilation de $fichier..."
       cc -c $fichier
done

users=`who | cut -f 1 -d ' ' | uniq`
for user in $users;
do
       echo "L'utilisateur $user est connecté"
done
```

➔ while / until

Syntaxes:

```
while/until expressions
do
       commandes
done
```

Exemples:

```
i=0
while test $i -lt 10
do
    echo "passe numéro $i"
    i=$(( $i + 1 ))
done

until [ "$saisie" = "stop" -o "$saisie" = "quit" ]
```



```
do
    read saisie
    echo "mot: $saisie"
done
```

➔ case

Syntaxe:

```
case expressions in
expression ) commandes ;;
expression ) commandes ;;
expression ) commandes ;;
...
* ) commandes ;;
esac
```

Exemple:

```
for param in $*;
do
       case $param in
       -f )
              echo "option -f utilisée"
              ;;

       plus )
              echo "argument plus"
              ;;
       * )
              echo "paramètre inconnu: $param"
       esac
done
```

Commandes internes

Un certain nombre de commandes sont exécutées directement par le shell et ne sont pas des programmes externes. Certaines correspondent à des programmes existant (comme `kill`, pour certains shells). Voici les plus utiles.

➔ read

Syntaxe:

```
read variables
```

Stocke dans les variables les mots tapés au clavier par l'utilisateur.

Exemple:

```
read mot1 mot2
echo "L'utilisateur a tapé $mot1 suivit de $mot2"
```

➔ exit

Syntaxe:

```
exit valeur
```

Termine le script immédiatement. Valeur est la valeur de retour du script (0 par défaut).

Exemple:

```
if [ "$sortie" = "oui" ]
then
       exit 1
fi
```

➔ eval

Syntaxe:

```
eval commandes
```

Exécute une liste de commandes. Fonctionne comme les guillements inversés ``.

Commandes externes

➔ test

Permet d'évaluer des expressions booléennes. Il y a trois types de tests:

- Tests sur les fichiers.
- Comparaisons de chaînes de caractères.
- Comparaisons de nombres.

Tous les tests sont expliqués dans la page du manuel UNIX de la commande `test`.

En voici quelques-uns parmi les plus utilisés:

-d fichier	Vrai si le fichier existe et est un répertoire
-f fichier	Vrai si le fichier existe et test un fichier normal
-e fichier	Vrai si le fichier existe quel que soit sont type
-z chaine	Vrai si la chaîne est vide
s1 = s2	Vrai si la chaîne s1 est égale à la chaîne s2
s1 != s2	Vrai si la chaîne s1 est différente de la chaîne s2
n1 -eq n2	Vrai si le nombre n1 est égal au nombre n2
n1 -ne n2	Vrai si le nombre n1 est différent du nombre n2
n1 -gt n2	Vrai si le nombre n1 est plus grand que n2
n1 -lt n2	Vrai si le nombre n1 est plus petit que n2

En plus de ces expressions, on peut utiliser les opérateurs suivants:

! expr	NON booléen
expr -a expr	ET booléen
expr -o expr	OU booléen

Les expressions peuvent êtres entourées de parenthèses.

Exemples:

```
if test -f /etc/rc.conf
then

while test "$i" -gt 10
do
```

La commande test a un deuxième nom: [(*crochet*), permettant une syntaxe plus claire dans les scripts. Attention aux espaces après le [et avant le] avec cette notation.

Exemple:

```
if [ ( "$rep" = "oui" ) -o ( "$rep" = "yes" ) ]
then
...
```

➔ expr

Sert à évaluer des expressions, et notamment les expressions arithmétiques.

Les opérateurs utilisables sont:

Opérations booléennes: | &

Opérations arithmétiques: + - * / %

Opérateurs de comparaisons: = != < > <= >=

Parenthèses: ( )

Note: les caractères (,) et * doivent être précédés de \ (*antislash*).

Exemples:

```
expr 1 + 1
expr \( 1 + 1 \) \* 2
a=`expr $a + 1`
```

Un autre manière d'évaluer une expression arithmétique dans un script est d'utiliser la notation suivante:

```
$(( expressions ))
```

Exemple:

```
a=$(( $a + 1 ))
```

➔ sleep

Cette commande n'est pas spécifique aux scripts mais est très utile. Elle sert à attendre un certain nombre de secondes.

Exemple:

```
sleep 60
```

