Administration des systèmes d'exploitation
Linux – Les scripts et les variables

1 Les Scripts

On appelle shell-script un fichier texte exécutable (droit x) qui contient des commandes exécutables par le shell. Sous Windows, on parle plutôt de « batch » ou de « traitement par lot ».

Le script peut être un simple enchainement de commandes linux, ou plus élaboré, en faisant appel à des variables, des tests, des itérations, …

Un shell-script est parfaitement adapté pour les opérations de maintenance sur un serveur Linux.

2 Les variables

On appelle **variable** une zone de stockage en mémoire vive. L'expression « variable » indique que son contenu peut être modifié à tout instant par un utilisateur ou un programme.
Une variable est définie par son NOM, son CONTENU, son ADRESSE MEMOIRE.

Dès qu'un shell-script doit exécuter des tâches complexes, on aura besoin d'utiliser des variables pour stocker temporairement des informations.

Le shell permet de créer des variables mémoires. Il suffit de leur attribuer un nom quelconque ; le shell fera tout seul l'association entre ce nom et l'adresse mémoire correspondante. En programmation Shell, on gère les variables par leur nom, et pas par leur adresse mémoire.

Il existe plusieurs types de variables :

- système …définies par le Shell (variables du système, ex : $HOME)
- utilisateur …que vous créez vous même
- spéciale …en lecture seule, gérée par le Shell (ex : $#)

Contenu d'une variable : des octets représentant des nombres ou des caractères alphanumériques.
La longueur est théoriquement non limitée. La longueur n'est pas constante.

2.1 La commande echo :

La commande ECHO sert à afficher du texte dans un script, mais aussi à afficher le contenu d'une variable.
Avec l'option –e, la commande **echo** admet les paramètres suivants permettant une mise en forme:

\a	alarme sonore
\n	saut de ligne
\b	retour arrière
\t	marque de tabulation
\c	fin de la sortie et annulation saut de ligne
\\	affichage d'un backslash
\xxx	indication d'un nombre octal (pour caractères spéciaux)

Exemple : `echo -n "bonjour \n Albert"`

Avec l'option –n , la commande **echo** n'effectue pas de retour à la ligne automatique.

2.2 Les Variables système

Elles sont prédéfinies par le système et écrites en majuscules. (voir commande **set** ou **env**)
Signification des variables : se reporter au manuel bash (man bash).

Pour visualiser le contenu d'une variable : **echo** $NOM_VARIABLE

Exemple : ***echo $PATH*** affiche la liste des dossiers dans lesquels le shell recherche les commandes tapées au clavier.

2.3 Créer vos variables :

Syntaxe à utiliser :

> **variable=valeur** (sans espaces !)

2.3.1 Règles de constitution d'un nom :

Le nom est constitué par des lettres minuscules et majuscules, des chiffres et le caractère soulignement.
Contrainte : pas de chiffre en tête, ni de soulignement.
Convention : nom des variables utilisateur en minuscules.
Variables contenant des caractères spéciaux (comme espace ..etc). Deux possibilités :
- encadrer le texte entre guillemets ou apostrophes :

Exemple : **variable="à nouveau moi"**

- utiliser le caractère d'échappement :

Exemple : **variable=De\ nouveau\ moi**

2.3.2 Calculer avec une variable

Si la variable n'est pas préalablement déclarée (voir paragraphes suivants), on peut demander au système d'évaluer une expression en utilisant les symboles `$(( expression ))`

Exemple :
```
a=3
b=$((a * 3))
echo $b
```
Résultat : 9

2.3.3 Déclarer une variable :

La déclaration d'une variable n'est pas nécessaire comme dans un langage évolué (C++).
La commande **declare** (voir le *man bash*) permet de préciser l'utilisation qui sera faite de la variable.

Par exemple :

Variable strictement numérique

Il est possible d'affecter à une variable le type **integer** (entier) à l'aide de la commande **declare.** Cela est utile si cette variable est utilisée dans des calculs : la syntaxe sera plus simple.

Exemple :
```
declare -i nombre
nombre=3*9
echo $nombre
```
Résultat : 27

```
nombrebis=3*9
echo $nombrebis
```
Résultat : 3*9 Ce n'est pas un nombre !!

Nb : L'affectation d'une chaîne de caractères à cette variable conduit à une valeur systématiquement nulle.
Lors de l'affectation d'une valeur à une variable de type **integer**, on peut utiliser différents opérateurs (Cf. annexe)

Variable en lecture seule :

Protéger une variable contre les modifications : **declare -r variable**=valeur
Nb : La commande **readonly** effectue la même action).

Exemple :
```
declare -r ma_variable=987
```

Une variable verrouillée ne pourra pas être détruite par la commande `unset`. Son contenu ne pourra être modifié. D'où l'importance de fixer sa valeur lors de la déclaration.

Visualiser les variables verrouillées : **declare -r** (sans arguments)

2.3.4 *Supprimer une variable :UNSET*

unset

Exemple : ***unset var***

2.3.5 *Saisir une variable au clavier :READ*

La commande **read** permet de saisir des variables à partir de l'entrée standard (le clavier par exemple).

Exemple :

```
read -p "Quel est votre nom: " nom
echo Vous avez dit : $nom
```

```
Quel est votre nom: Jean <ENTREE>
Vous avez dit : Jean
```

2.3.6 *Affecter à une variable le résultat d'une commande shell :*

Lorsqu'une commande est placée entre parenthèses suivant un $, cette commande est remplacée par les sorties que fait la commande sur le canal standard.
Autre notation possible : placer la commande entre simples cotes inverses ` `

Exemple :

```
ici=$(pwd)   ou    ici=`pwd`
echo Vous etes dans le dossier $ici
```

L'expression entre parenthèses (ou entre simple cotes) peut être complexe et associer des tubes (pipe |).

2.4 Variables spéciales :

Elles ne sont accessibles qu'en lecture, elles sont gérées par le shell

$?	valeur de retour de la dernière commande exécutée
$$	numéro du processus en cours (PID)
$!	numéro du dernier processus en arrière plan
$-	options du shell
$0	premier argument (nom de la commande shell)
$1	2eme argument
……	
$9	10eme argument
$*	tous les arguments
$@	tous les arguments sous forme séparée
$#	Nombre d'arguments passés à la commande shell

2.4.1 *Arguments transmis à une commande ou à un script:*

Lors de l'appel d'un script, il est possible de transmettre des paramètres sous forme d'arguments dans la ligne de commande. Les arguments sont les mots placés après le nom de la commande :

Exemple : `ls -l`
la commande est *ls* et l'argument est `-L`

`vi toto`
la commande est *vi* et l'argument est *toto*

NB : La commande elle-même est considérée comme un argument

Le premier paramètre sera repéré par $1, le second par $2 ..etc jusqu'au 9ème.
La variable $0 contient le nom du script.

❑ Exemple avec le script suivant (fichier *essai6.bash*) :

```
#!/bin/bash
echo "Script : $0"
echo "Paramètres : 1= $1 2= $2 3= $3"
echo "Nombre de paramètres : $#"
#Fin du script
```

Exécuter le script avec des arguments : `essai6.bash bonjour 50 linux`
Donne le résultat :

```
Script : essai6.bash
Paramètres : 1= bonjour 2= 50 3= linux
Nombre de paramètres : 4
```

2.4.2 Valeur retournée

Chaque commande retourne une valeur (*exit status*)
Cette valeur indique si une erreur s'est produite au cours de l'exécution (valeur retournée différente de zéro) ou non (valeur retournée égale à zéro).
La valeur retournée par la dernière commande est stockée dans la variable spéciale **$?** (Cf. variables spéciales dans le document annexe)

Exemple :
*Un fichier nommé **fleurs.txt** contenant les mots suivants :*

tulipe
rose
narcisse

Résultat des commandes suivantes :

```
grep primevere fleurs.txt
echo $?                          Affiche : 1  donc la commande a échoué

grep rose fleurs.txt
echo $?                          Affiche : 0  donc la commande a réussi
```

2.5 Extension de variables à l'aide d'accolades.

Pour que le shell distingue le nom des variables du texte environnant, il faut placer les variables entre accolades.
Exemple : script de duplication du fichier fichier1 en fichier2

```
a=fichier
cp ${a}1 ${a}2
```

2.6 Détermination de la longueur d'une variable.

La commande **${#variable}** donne le nombre de caractères contenus dans la variable.

Exemple :

```
a=abracadabra
echo "Le mot $a contient ${#a} caractères"
```

3 Temporisation :

sleep suspend l'exécution du shell script pendant n secondes.

Exemple :

```
sleep 10
```

4 Evaluation d'expressions conditionnelles

4.1 La commande `test` ou `[…]`

Permet d'effectuer des tests sur les fichiers ou des variables.
Un test (en informatique) ne sait renvoyer que 2 valeurs : VRAI ou FAUX.

Ce résultat doit être ensuite associé à une commande de type « si le test est vrai, faire ceci, sinon faire cela »

On utilise la commande ***if*** ou les commandes **&&** et **||** (voir paragraphes suivants) pour dire quoi faire en fonction du résultat du test.

Exemple : Ces 4 commandes donnent le même résultat :

```
test -e /home/toto &&  echo Le fichier toto existe
```

```
[ -e /home/toto ] && echo Le fichier toto existe
```

```
if [ -e /home/toto ]
then
      echo Le fichier toto existe
fi
```

```
if test -e /home/toto
then
      echo Le fichier toto existe
fi
```

A vous de choisir votre écriture préférée.

Quelques tests utiles :

Tester un fichier qui s'appelle `fic`

`[ -e fic ]` *(vrai si fic existe)*
`[ -f fic ]` *(vrai si fic est un fichier normal)*
`[ -d fic ]` *(vrai si fic est un dossier)*
`[ -r fic ]` *(vrai si fic a l'autorisation d'accès en lecture)*
`[ -w fic ]` *(vrai si fic a l'accès en écriture)*
`[ -x fic ]` *(vrai si fic a accès en exécution)*
`[ -s fic ]` *(vrai si fic est non vide)*

Tester une variable contenant du texte ($a par ex.) :

`[ -z "$a" ]` *(vrai si $a est vide)*
`[ -n "$a" ]` *(vrai si $a est non vide)*
`[ "$a" = toto ]` *(vrai si $a $a contient le mot toto)*
`[ "$b" != tintin ]` *(vrai si $a $b ne contient pas le mot tintin)*

Tester une variable numérique (idem) :

test "$nombre" -eq 5 *(equal)*
.... -ne 4 *(not equal)*
.... -lt 5 *(less than)*
.... -gt 4 *(greater than)*
.... -le 3 *(less or equal)*
.... -ge 2 *(greater or equal)*

Combinaison de tests :

! *(négation du test)*
-a *ET logique*
-o *OU logique*

4.2 Enchaînement conditionnel de commandes :

Le shell permet d'enchaîner des commandes de manière conditionnelle.
Avec le ET logique (&&), la deuxième commande ne s'exécute que si la première a réussi.
Avec le OU logique (||), la deuxième commande ne s'exécute que si la première a échoué.

Exemple :

```
[ -r fic ] && echo "vous pouvez lire ce fichier"
[ -s fic ] || echo "ce fichier est vide"
```

5 Les structures de contrôle

5.1 Les conditions (if, case)

if

```
if [ -f fichier ]
then  {
      echo "c'est un fichier ordinaire"
}
else  {
      echo "C'est peut-être un dossier"
}
fi
```

case

```
case var in
      cas1 ) cde1;;
      cas2 ) cde2;;
      cas3) cde3;;
      *) cde4;
esac
```

Exemple :

```
jour=$(date +%d%m)
case $jour in
  01/07 ) echo "C'est les vacances";;
  01/09) echo "C'est la rentrée";;
  *) echo "Rien à signaler !"
esac
```

5.2 Les itérations : boucles *for, while*

for : boucle sur une liste de variable

structure :

```
for var in liste
 do
      commande
 done
```

Exemple :

```
for a in 1 2 3 4
do
      cp fic fic$a
done
```

Ici la variable a prendra successivement toutes les valeurs de la liste.
si on écrit for var in $@ la liste fournie sera les arguments passés par la ligne de commande.

while : Boucle tant qu'une condition est vraie

structure :

```
while expression
do
      commande
done
```

Exemple :

Version avec declare

```
declare -i n
n=0

while [ $n -lt 10 ]
do
      echo $n
      n=$n+1
done
```

Version avec evaluation $(())

```
n=0

while [ $((n)) -lt  10 ]
do
      echo $n
      n=$((n+1))
done
```

Variante du FOR : Façon C++, boucle de comptage.

Cette variante est à manipuler avec précaution car elle ne respecte pas clairement la syntaxe du bash.
Préférez l'utilisation de la boucle WHILE.

```
for ((e1;e2;e3))
do
    commandes
done
```

e1 : état de départ.
e2 : expression évaluée pour quitter la boucle : tant que l'évaluation est VRAIE, on continue.
e3 : opération à effectuer à chaque passage dans la boucle

```
for ((i=0 ; i<10 ; i++))
do
    echo $i
done
```

break, continue

L'instruction **Break** permet de forcer la sortie d'une boucle.
continue permet de passer à la Nième itération.

Exemple :

```
while true
do
        echo "donner un nom de fichier \c"
        read fich
                if [ -z "$fich" ]
                then
                        break
                fi
        echo "doit on supprimer $fich ?\c"
        read rep
        if [z "$rep" -o $rep = "NON"]
        then
                continue
        else
                rm $fich
        fi
done
```

5.3 Divers

select

C'est une instruction bash qui permet la création de menus.

Structure :

```
select var in liste_var
do
        commandes
done
```

Exemple :

```
PS3="Numéro : "
select nom in david marie michel michelle annie sylviane
do
        if [ -n "$nom"]
        then
                echo "$nom a été choisi"
                break;
        fi
done
```

```
linux-htc2:~ # ./selectest
1) david
2) marie
3) michel
4) michelle
5) annie
6) sylviane
Numero :2
marie a ete choisi
linux-htc2:~ #
```

6 Les fonctions

Le shell permet de définir ses propres fonctions à l'intérieur d'un script.

structure :

```
nom_fonction()
{
      cdes
}
```

Exemple :

```
fin()
{
      clear
      echo "au revoir "
      exit
}
```

Si on voulait écrire une commande ***dir*** affichant les noms des sous-dossiers du dossier en cours :

```
dir()
{
      clear
      echo "Répertoire courant : $(pwd)"
      for w in $(ls)
      do
            if [ -d $w ]
            then
                  echo $w
            fi
      done
}
```

7 ANNEXES

Variables spéciales :
elles ne sont accessibles qu'en lecture, elles sont gérées par le shell

$?;valeur de retour de la dernière commande exécutée
$$ numéro du process en cours (PID)
$! numéro du dernier process en arrière plan
$- option du shell
$0 premier paramètre (nom de la commande)
$1 2eme paramètre
……
$9 10eme paramètre
$* tous les paramètres
$@ tous les paramètres sous forme séparée
$# Nombre de paramètres passé à la commande

Opérateurs sur les variables entières

+ - * /	opérations de base sur les variables entières (integer)
%	modulo reste de la division entière (integer)
<> >= <=	comparaison, la valeur retournée sera 1 si vrai 0 si faux
== !=	égalité et inégalité
&& ||	comparaison avec le ET logique ou le OU logique
& | ^	comparaison bit à bit de la variable &(ET), | (OU), ^ (OU exclusif)

Rappel :
valeur 0 = vrai = true
valeur 1 = faux = false

Manipulation des chaines de caractères :

Voir le site : http://abs.traduc.org/abs-5.0-fr/ch09s02.html

Exemple :
Extraction de caractères dans une variable contenant une chaine de caractères :
Syntaxe : **${variable<opérateur>critère_de_recherche}**

Critère de recherche : chaine_1*chaine_2 le texte compris entre la chaîne_1 et la chaine_2

#	recherche de la plus petite occurrence du critère de recherche au début du texte
##	recherche de la plus grande occurrence du critère de recherche au début du texte
%	recherche de la plus petite occurrence du critère de recherche à la fin du texte
%%	recherche de la plus grande occurrence du critère de recherche à la fin du texte

Exemple :
```
var="Le chêne et le roseau au poteau"
echo ${var#L*o}
```
Résultat : `seau au poteau`

```
echo ${var##L*o}
```
Résultat : `teau`