Les variables

On en distingue trois types : utilisateur, système et spéciales. Le principe est de pouvoir affecter un contenu à un nom de variable, généralement une chaîne de caractère ou des valeurs numériques.

1. Nomenclature

Un nom de variable obéit à certaines règles :

- Il peut être composé de lettres minuscules, majuscules, de chiffres, de caractères de soulignement.
- Le premier caractère ne peut pas être un chiffre.
- Le taille d'un nom est en principe illimitée (il ne faut pas abuser non plus).
- Les conventions veulent que les variables utilisateur soient en minuscules pour les différencier des variables système. Au choix de l'utilisateur.

2. Déclaration et affectation

Une variable est déclarée dès qu'une valeur lui est affectée. L'affectation est effectuée avec le signe **=**, sans espace avant ou après le signe.

```
var=Bonjour
```

3. Accès et affichage

Vous accédez au contenu d'une variable en plaçant le signe **$** devant le nom de la variable. Quand le shell rencontre le **$**, il tente d'interpréter le mot suivant comme étant une variable. Si elle existe, alors le $nom_variable est remplacé par son contenu, ou par un texte vide dans le cas contraire. On parle aussi de référencement de variable.

```
$ chemin=/tmp/driss
$ ls $chemin
...
$ cd $chemin
$ pwd
/tmp/driss
$ cd $chemin/rep1
$ pwd
/tmp/driss/rep1
```

Pour afficher la liste des variables, on utilise la commande **env**. Elle affiche les variables utilisateur et les variables système, nom et contenu.

```
$ env
LESSKEY=/etc/lesskey.bin
NNTPSERVER=news
INFODIR=/usr/local/info:/usr/share/info:/usr/info
MANPATH=/usr/local/man:/usr/share/man
KDE_MULTIHEAD=false
SSH_AGENT_PID=26377
HOSTNAME=p64p17bicb3
DM_CONTROL=/var/run/xdmctl
XKEYSYMDB=/usr/share/X11/XKeysymDB
HOST=p64p17bicb3
SHELL=/bin/bash
TERM=xterm
PROFILEREAD=true
HISTSIZE=1000
...
```

Une variable peut contenir des caractères spéciaux, le principal étant l'espace. L'exemple suivant ne fonctionne pas :

```
$ c=Salut les copains
les: not found
$ echo $c
```

Pour cela il faut soit verrouiller les caractères spéciaux un par un, soit les mettre entre guillemets ou apostrophes.

```
c=Salut\ les\ Copains # Solution lourde
c="Salut les copains" # Solution correcte
c='Salut les copains' # Solution correcte
```

La principale différence entre les guillemets et les apostrophes est l'interprétation des variables et des substitutions. " et ' se verrouillent mutuellement (voir dernier exemple).

```
$ a=Jules
$ b=Cesar
$ c="$a $b a conquis la Gaule"
$ d='$a $b a conquis la Gaule'
$ echo $c
Jules Cesar a conquis la Gaule
$ echo $d
$a $b a conquis la Gaule
$ echo "Linux c'est top"
Linux c'est top
$ echo 'Linux "trop bien"'
Linux "trop bien"
```

4. Suppression et protection

Vous supprimez une variable avec la commande **unset**. Vous protégez une variable en écriture et contre sa suppression avec la commande **readonly**. Une variable en lecture seule, même vide, est figée. Il n'existe aucun moyen de la replacer en écriture et de la supprimer, sauf en quittant le shell.

```
$ a=Jules
$ b=Cesar
$ echo $a $b
Jules Cesar
$ unset b
$ echo $a $b
Jules
$ readonly a
$ a=Neron
a: is read only
$ unset a
a: is read only
```

5. Export

Par défaut une variable n'est accessible que depuis le shell où elle a été définie. La variable **a** est déclarée sous l'invite du shell courant puis est affichée par un script lancé depuis ce même shell. Ce dernier ne connaît pas la variable **a** : rien ne s'affiche.

```
$ a=Jules
$ echo 'echo "a=$a"' > voir_a.sh
$ chmod u+x voir_a.sh
$ ./voir_a.sh
a=
```

La commande **export** permet d'exporter une variable de manière à ce que son contenu soit visible par les scripts et autres sous-shells. Les variables exportées peuvent être modifiées dans le script, mais ces modifications ne s'appliquent qu'au script ou au sous-shell. Cette fois le premier script peut accéder à la variable **a** exportée. Mais les modifications restent locales au script. Une fois celui-ci terminé la modification disparaît.

```
$ a=Jules
$ echo 'echo "a=$a"' > voir_a.sh
$ chmod u+x voir_a.sh
$ export a
$ ./voir_a.sh
a=Jules
$ echo 'a=Neron ; echo "a=$a"' >> voir_a.sh
$ ./voir_a.sh
a=Jules
a=Neron
$ echo $a
Jules
```

6. Accolades

Les accolades de base { } permettent d'identifier le nom d'une variable. Imaginez la variable fichier contenant le nom de fichier 'liste'. Vous voulez copier liste1 en liste2.

```
$ fichier=liste
$ cp $fichier1 $fichier2
cp: opérande fichier manquant
Pour en savoir davantage, faites: « cp --help ».
```

Cela ne fonctionne pas car ce n'est pas $fichier qui est interprété mais $fichier1 et $fichier2 qui n'existent pas.

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

Dans ce cas, cette ligne équivaut à :

```
$ cp liste2 liste1
```

7. Accolades et remplacement conditionnel

Les accolades disposent d'une syntaxe particulière.

{variable:Remplacement}
Selon la valeur ou la présence de la variable, il est possible de remplacer sa valeur par une autre.

Remplacement	Signification
{x:-texte}	Si la variable x est vide ou inexistante, le texte prendra sa place. Sinon c'est le contenu de la variable qui prévaudra.
{x:=texte}	Si la variable x est vide ou inexistante, le texte prendra sa place et deviendra la valeur de la variable.
{x:+texte}	Si la variable x est définie et non vide, le texte prendra sa place. Dans le cas contraire une chaîne vide prend sa place.
{x:?texte}	Si la variable x est vide ou inexistante, le script est interrompu et le message texte s'affiche. Si texte est absent un message d'erreur standard est affiché.

```
$
echo $nom
```

```
$ echo ${nom:-Jean}
Jean
$ echo $nom

$ echo ${nom:=Jean}
Jean
$ echo $nom
Jean
$ echo ${nom:+"Valeur définie"}
Valeur définie
$ unset nom
$ echo ${nom:?Variable absente ou non définie}
nom: Variable absente ou non définie
$ nom=Jean
$ echo ${nom:?Variable absente ou non définie}
Jean
```

8. Variables système

En plus des variables que l'utilisateur peut définir lui-même, le shell est lancé avec un certain nombre de variables prédéfinies utiles pour certaines commandes et accessibles par l'utilisateur. Le contenu de ces variables système peut être modifié mais il faut alors faire attention car certaines ont une incidence directe sur le comportement du système.

Variable	Contenu
HOME	Chemin d'accès du répertoire utilisateur. Répertoire par défaut en cas d'utilisation de CD.
PATH	Liste de répertoires, séparés par des : où le shell va rechercher les commandes externes et autres scripts et binaires. La recherche se fait dans l'ordre des répertoires saisis.
PS1	Prompt String 1, chaîne représentant le prompt standard affiché à l'écran par le shell en attente de saisie de commande.
PS2	Prompt String 2, chaîne représentant un prompt secondaire au cas où la saisie doit être complétée.
IFS	Internal Field Separator, liste des caractères séparant les mots dans une ligne de commande. Par défaut il s'agit de l'espace, de la tabulation et du saut de ligne.
MAIL	Chemin et fichier contenant les messages de l'utilisateur.
SHELL	Chemin complet du shell en cours d'exécution.
LANG	Définition de la langue à utiliser ainsi que du jeu de caractères.
USER	Nom de l'utilisateur en cours.
LOGNAME	Nom du login utilisé lors de la connexion.
HISTFILE	Nom du fichier historique, généralement $HOME/.sh_history.
HISTSIZE	Taille en nombre de lignes de l'historique.
OLDPWD	Chemin d'accès du répertoire accédé précédemment.
PS3	Définit l'invite de saisie pour un select.
PWD	Chemin d'accès courant.
RANDOM	Génère et contient un nombre aléatoire entre 0 et 32767.

9. Variables spéciales

Il s'agit de variables accessibles uniquement en lecture et dont le contenu est généralement contextuel.

Variable	Contenu
$?	Code retour de la dernière commande exécutée.
$$	PID du shell actif.
$!	PID du dernier processus lancé en arrière-plan.
$-	Les options du shell.

```
$ echo $$
23496
$ grep memoire liste
$ echo $?
1
$ grep souris liste
souris  optique 30      15
$ echo $?
0
$ ls -lR >toto.txt 2>&1 &
26675
$ echo $!
26675
```

le pid est un identifiant des processus en cours

$ps -aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.0 176832 5260 ? Ss 05:48 0:01 /sbin/init

root 2 0.0 0.0 0 0 ? S 05:48 0:00 [kthreadd]

root 3 0.0 0.0 0 0 ? S 05:48 0:00 [ksoftirqd/0]

root 5 0.0 0.0 0 0 ? S< 05:48 0:00 [kworker/0:0H]

....

on souhaite arrêter un processus : *kill PID*

```
$ kill 215
```

10. Longueur d'une chaîne

Il est possible d'obtenir la longueur d'une chaîne avec le caractère #.

```
$ a=Jules
$ echo "Longueur de $a : ${#a}"
Longueur de Jules : 5
```

11. Tableaux et champs

Deux moyens sont disponibles pour déclarer un tableau, l'un avec l'utilisation des crochets **[ ]**, l'autre avec la création globale. Le premier élément est 0 le dernier 1023. Pour accéder au contenu du tableau il faut mettre la variable ET l'élément entre accolades **{ }**.

```
$ Nom[0]="Jules"
$ Nom[1]="Romain"
$ Nom[2]="Francois"
$ echo ${Nom[1]}
Romain
```

ou :

```
$ Nom=(Jules Romain Francois)
$ echo ${Nom[2]}
Francois
```

Pour lister tous les éléments :

```
$ echo ${Nom[*]}
Jules Romain Francois
```

Pour connaître le nombre d'éléments :

```
$ echo ${#Nom[*]}
3
```

Si l'index est une variable, on ne met pas le $ devant celui-ci :

```
$ idx=0
$ echo ${Nom[idx]}
Jules
```

12. Variables typées

Les variables peuvent être typées en entier (integer) avec la commande **typeset -i** le permet. L'avantage est qu'il devient possible d'effectuer des calculs et des comparaisons sans passer par expr. La commande **let** ou **((...))** permet des calculs sur variables.

Opérateur	Rôle
+ - * /	Opérations simples
%	Modulo
< > <= >=	Comparaisons, 1 si vraie, 0 si faux
== !=	Égal ou différent
&& ||	Comparaisons liées par un opérateur logique
& | ^	Logique binaire AND OR XOR

```
$ typeset -i resultat
$ resultat=6*7
$ echo $resultat
42
$ resultat=Erreur
ksh: Erreur: bad number
$ resultat=resultat*3
126
$ typeset -i add
$ add=5
$ let resultat=add+5 resultat=resultat*add
$ echo $resultat
50
```

13. Alias

Un alias est un raccourci d'une commande avec d'éventuels paramètres. Il se définit avec la commande **alias**. Utilisée seule elle liste les alias disponibles.

```
$ alias
alias ..='cd ..'
alias ...='cd ../..'
alias cd..='cd ..'
alias dir='ls -l'
alias l='ls -alF'
alias la='ls -la'
alias ll='ls -l'
alias ls='ls $LS_OPTIONS'
alias ls-l='ls -l'
alias md='mkdir -p'
alias o='less'
alias rd='rmdir'
...
```

Vous pouvez créer vos propres alias.

```
$ alias deltree='rm -rf'
```

14. Groupement de commandes

Le chaînage de commande est possible avec « ; ». Il est aussi possible de grouper les commandes. Quand vous exécutez les commandes suivantes :

```
$ uname -a ; pwd ; ls -l >resultat.txt &
```

Seule la dernière commande est exécutée en tâche de fond et seul son résultat est redirigé dans le fichier resultat.txt. Une solution serait :

```
$ uname -a >resultat.txt & ; pwd >>resultat.txt & ; ls -l >>resultat.txt &
[1] 18232
[2] 18238
[3] 18135
```

C'est une solution complexe et qui ne fonctionnera pas toujours. De plus même si les commandes sont lancées séquentiellement, elles tournent toutes en parallèle et c'est la première finie qui écrira en premier dans le fichier. La solution consiste en l'utilisation des parenthèses.

```
$ (uname -a ; pwd ; ls -l) > resultat.txt &
[1] 18239
$
[1]    Done                 ( uname -a; pwd; ls -l ) > resultat.txt
```

Dans ce cas, toutes les commandes placées entre les parenthèses sont lancées par un sous-shell, qui va ensuite exécuter les commandes précisées séquentiellement telles qu'indiquées. Ainsi la redirection concerne l'ensemble des commandes et rien n'empêche de lancer ce sous-shell en arrière-plan. Vous distinguez bien d'ailleurs un seul PID 18239 lors de l'exécution des commandes.

Une seconde possibilité est l'utilisation des accolades {...}. Dans ce cas aucun sous-shell n'est exécuté, et si une commande interne (cd ou autre) est exécutée, elle concerne le shell actif. L'accolade fermante doit être placée juste après un ;.

```
$ { uname -a; pwd; ls -l; } > resultat.txt
```

Vous pouvez faire facilement la différence entre les deux syntaxes avec exit. Le premier exemple semble ne rien faire, alors qu'il quitte le shell fils. Le second sort de votre shell.

```
$ (exit)
$ { exit; }
```

Attention avec les parenthèses, notamment en programmation. Comme le groupement est lancé au sein d'un autre processus, les éventuelles variables

modifiées au sein du groupement ne seront pas visibles une fois l'exécution terminée.

15. Liaison et exécution conditionnelle

En plus du chaînage classique, les commandes peuvent être liées et exécutées de façon conditionnelle. La condition d'exécution d'une commande est la réussite ou non de la précédente. Chaque commande une fois exécutée renvoie un code de retour, généralement 0 si tout s'est bien passé, 1 ou 2 en cas d'erreur. Le shell peut récupérer cette valeur par la variable $?.

```
$ ls
...
$ echo $?
0
```

Les caractères **&&** et **||** permettent d'effectuer une exécution conditionnelle.

```
commande1 && commande2
commande1 || commande2
```

La commande située après **&&** sera exécutée uniquement si la commande précédente a retourné 0 (réussite). La commande située après **||** ne sera exécutée que si la commande précédente a retourné autre chose que 0.

```
$ grep "souris" liste && echo "Souris trouvee" || echo "Souris introuvable"
souris  optique 30      15
Souris trouvee
$ grep "memoire" liste && echo "Memoire trouvee" || echo "Memoire
introuvable"
Memoire introuvable
```

EXERCICES

Exécution conditionnelle

1- Supprimez un fichier fic. Si la suppression a réussi affichez OK. Sinon affichez ERREUR et faites un ls - l de ce fichier.

Variables

1- Déclarez une variable VAR contenant votre nom.

2- Une variable a contient la valeur 2. Lancez depuis la même console un second bash. Affichez a. Que contient la variable a ?

3- Créez une variable b contenant 3 et exportez-la. Lancez un bash fils et incrémentez b. Sortez et affichez b. Que contient-elle ?

- A - Rien.

- B - 4.
- C - 3 car b a été incrémentée dans un processus fils différent du père.
- D - 1 car b n'existe pas dans le fils, il a été remis à 0.

4- La variable C contenant du texte, qu'affiche echo ${C:+"coucou"} et que contient C ensuite ?

5- Modifiez le path par défaut en lui ajoutant /opt/bin en première position.

6- Comment connaître la valeur de retour d'une suppression de fichier ?

7- Donnez deux moyens d'additionner deux variables A et B contenant des entiers et de placer le résultat dans A.