

## Variables

Le shell distingue trois types de variables. Chacune a des caractéristiques propres :

- les variables définies par l'utilisateur
- les variables prédéfinies internes au Shell
- les variables prédéfinies internes par exécution de fichier

### Affectation :

Le shell offre la possibilité de travailler avec des variables. L'opération consiste à affecter un contenu appelé *valeur* (value) à une variable. La longueur d'une variable n'est pas constante mais elle ne peut recevoir que des *caractères* (characters). Le contenu de la variable peut-être interprété sous forme numérique ou de *chaînes de caractères* (string) et sont régies par les règles suivantes :

1. un nom de variable peut être composé de lettre minuscules/majuscules, de chiffres et du caractère de soulignement.
2. il n'est pas permis d'utiliser un chiffre comme premier caractère du nom d'une variable. Le caractère de soulignement est évité en première lettre.
3. la longueur du nom d'une variable n'a pas de limite théorique.
4. par convention, les noms de variables définies par l'utilisateur doivent être écrits en minuscules pour les différencier des variables système.

ex : `variable=valeur` affectation : forme effectivement un mot unique sans espace autour de "="

S'il est possible d'affecter un texte vide à une variable du shell, il faut distinguer cette variable définie/indéfinie d'une variable vide ou non :

ex : `variable=` la variable est définie mais vide (aucune affectation de chaîne de caractères)

Une variable peut contenir des caractères spéciaux, tel que l'espace, mais pour les dissimuler au shell, il faut mettre le texte entre des guillemets (" ") ou des apostrophes ( ' ') ou des backslashes (\ \) :

```
ex: my_text="Quel est le nom de la mascotte du système Linux ?"
    my_text='Quel est le nom de la mascotte du système Linux ?'
    my_text=Quel\ est\ le\ nom\ de\ la\ mascotte\ du\ système\ Linux\ \?
```

`echo` affiche le contenu d'une variable : la commande renvoie le texte par la sortie standard (stdout). Si le shell trouve la variable dans la liste des paramètres de la commande `echo` elle affiche le contenu de la variable précédée du signe \$.

```
ex: name=Tux
    os=Linux
    mascot="$name est le nom de la mascotte du système $os."
    echo $mascot affiche Tux est le nom de la mascotte du système Linux.
```

`unset` supprime une variable affectée ou non, sauf si elle est *verrouillée* (readonly) :

```
ex: unset name os suppression de plusieurs variables
```

`readonly` permet de définir une protection en écriture : le contenu de cette variable ne pourra plus être modifié, ni affecté de texte si elle est déclarée vide ou même être détruite. Une variable verouillée reste dans le shell jusqu'à la fin du processus.

ex : `readonly var` variable verouillée pour la compatibilité avec les autres shells

Les accolades `{` et `}` permettent de différencier le nom des variables du texte environnant : on les utilise pour mettre des noms de variables dans d'autres variables ou calculer le nombre de caractères d'une variable :

```
ex : user="/home/toto/stage"
    echo $user
    u1=$user1
    echo $u1 n'affiche pas le résultat escompté !
    u1=${user}1
    echo $u1 affiche le résultat escompté !
    /home/toto/stage1 résultat : affichage du contenu de la variable u1
```

L'opérateur `#` placé devant le nom de la variable et entre accolades `{}` permet de calculer le nombre de caractères contenu dans la variable :

```
ex : nb_car="texte pour calculer le nombre de caractères contenu dans la
variable" calcule le nombre de caractères du texte placé entre "...".
    echo "Le texte de la variable contient ${#nb_car} caractère(s)." affiche le
nombre de caractères contenu dans la variable nb_car.
```

### **Variables prédéfinies internes au shell**

Les variables prédéfinies sont, par convention, écrites en majuscules. Ce sont elles qui contiennent toutes les options par défaut de l'environnement de travail : le choix du shell, l'apparence (par défaut) du prompt, le type de terminal utilisé, le nom donné au serveur, le nombre de lignes et de colonnes de l'écran :

– les variables système : `$0`, `$n` et `$#`

Récupération des paramètres dans la ligne de commande :

`$0` nom de la commande

`$n` n pour nième paramètre

`$#` nombre de paramètres

`$*` liste de tous les paramètres

– les variables spéciales : `$?` et `$$`

`$?` retourne le code de la dernière la commande

`$$` le numéro de processus de la dernière commande

- variables d'environnement

HOME chemin d'accès au répertoire initial de l'utilisateur

PATH suite de chemins d'accès aux répertoires des exécutables

PS1 invite principale du shell en mode interpréteur

PS2 invite secondaire du shell en mode programmation

IFS séparateurs de champ des arguments

MAIL chemin d'accès à la boîte aux lettres utilisateur

MAILCHECK intervalle en sec au bout duquel le mail est contrôlé

CDPATH liste de chemins d'accès pour la commande cd

ENV nom du fichier des variables d'environnement

TERM nom du type de terminal

Les variables d'environnement servent à enregistrer des paramètres que les programmes peuvent lire ensuite. Elles sont désignées par un symbole \$ suivi de lettres, chiffres et symboles :

ex : la variable \$HOME est égale au répertoire *maison* (home) de l'utilisateur en cours en général /home/user

De même, la variable \$PATH représente le chemin de recherche que le shell va parcourir afin de trouver le fichier exécutable qui correspond à la commande que vous venez de taper :

ex : \$PATH=/bin:/usr/bin:/usr/local/bin

On peut afficher la valeur de ces variables par la commande echo, déjà abordée précédemment. Le nom de la variable doit être précédé du signe \$ :

ex : echo \$SHELL, \$LOGNAME, \$HOSTNAME, \$TERM, \$PATH

printenv ou env permet d'obtenir la liste de toutes ces variables prédéfinies :

ex : printenv affiche toutes les variables prédéfinies

env idem

set affiche les listes des variables définies dans le shell : les variables définies par l'utilisateur et les variables système :

ex : set affiche la liste des variables du shell