

Le Shell (B)

- A RACCOURCIS CLAVIER
- B EXECUTION DE TESTS
- C FONCTIONS
- D SUBSTITUTIONS
- E TYPES DE DONNEES
- F LONGUEUR DE CHAINE
- G TRAITEMENT DES CHAINES
- H OPERATEUR ..
- I TABLEAUX
- J COMMANDES
- K VARIABLES

A) Raccourcis clavier

TAB	Complète automatiquement un mot (commande ou argument), à partir de la position du curseur .
CTRL + a	déplace le curseur en début de ligne (équivalent à la touche origine).
CTRL + e	(end) déplace le curseur en fin de ligne (équivalent à la touche fin).
CTRL + p	(previous) rappelle la commande précédente (équivalent à la touche flèche vers le haut).
CTRL + n	(next) rappelle la commande suivante (équivalent à la touche flèche vers le bas).
CTRL + r	(research) rappelle la dernière commande contenant les caractères spécifiés .
CTRL + o	exécute la commande trouvée dans la recherche , et propose la suivante, dans le fichier historique.
CTRL + l	efface le contenu de l'écran (équivalent à la commande clear).
CTRL + u	efface le contenu de la ligne avant le curseur , et le place dans le presse-papier .
CTRL + k	efface le contenu de la ligne après le curseur et le place dans le presse-papier .
CTRL + w	efface le mot avant le curseur , et le place dans le presse papier .
CTRL + y	(yank)ajoute le contenu du presse-papier , à partir de la position du curseur .
CTRL + d	ferme le Shell courant (équivalent de la commande exit).
CTRL + c	envoie le signal SIGINT à la tâche au premier plan, ce qui a pour effet de l' interrompre .
CTRL + z	envoie le signal SIGSTP à la tâche au premier plan, ce qui a pour effet de la suspendre. Pour la réafficher on peut entrer : fg 'nom du processus'
CTRL + x CTRL + x	(car x est en forme de croix) alterne le curseur avec son ancienne position (<u>Attention</u> : Deux fois Ctrl + x).
CTRL + x CTRL + e	(<i>editor</i> , car reprend la variable \$EDITOR du shell) édite la ligne courante dans vi ou dans nano .
ALT + f	(forward) avance le curseur d'un mot.
ALT + b	(backward) recule le curseur d'un mot.

ALT + d

coupe le mot après le curseur.

ALT + u :transforme le mot courant en **majuscules** à partir du curseur, et se place à la fin du mot.**ALT + l**transforme le mot courant en **minuscules** à partir du curseur, et se place à la fin du mot.**ALT + c**met la lettre sous le curseur en **majuscule**, et se place à la fin du mot.**ALT + r**

annule les changements, et remet la ligne telle qu'elle était dans l'historique.

B) Exécution de tests

La commande **test** a déjà été étudiée.

Elle dispose d'un équivalent, plus récent et très utilisé : les **crochets ([])**.

Les expressions : **test expression** et **[expression]** sont identiques.

Exemples :

```
test -f /etc/passwd
echo $?
0
[ -f /etc/passwd ]
echo $?
0
```

Les opérateurs de test *historiques* ont été vus dans le support précédent.

Il existe également, pour les **fichiers** :

-h	Vrai si le fichier est de type lien physique
-L	Vrai si le fichier est de type lien symbolique
-b	Vrai si le fichier est de type spécial bloc
-c	Vrai si le fichier est de type spécial caractère
-p	Vrai si le fichier est de type tube nommé
-S	Vrai si le fichier est de type socket
-u	Vrai si le fichier dispose du SUID
-g	Vrai si le fichier dispose du SGID
-k	Vrai si le fichier dispose du Sticky Bit

et il existe, en plus, apparus avec le **Korn-Shell** :

fic1 -nt fic2	Vrai si le fichier fic1 est plus récent que fic2
fic1 -ot fic2	Vrai si le fichier fic1 est plus ancien que fic2
fic1 -ef fic2	Vrai si les fichiers ont la même inode
-O fic1	Vrai si l' utilisateur est propriétaire du fichier
-G fic1	Vrai si l' utilisateur appartient au groupe prop. du fichier

Commande :**[[]]**

Les **doubles-crochets** permettent également de **tester une expression**.

Avantage : Les guillemets autour des noms de variables ne sont plus obligatoires.

Inconvénient : Certains opérateurs changent, d'où de nombreuses sources de confusions.

Exemple : La commande :

```
if [ -w $fic -a \(-d $rep1 -o -d $rep2 \) ]  
...  
...  
if [[ -w $fic && ( -d $rep1 || -d $rep2 ) ]]  
...
```

est équivalente à la commande :

```
if [[ -w $fic && ( -d $rep1 || -d $rep2 ) ]]  
...
```

C) Fonctions

Les fonctions servent à **regrouper des commandes**, lorsqu'elles ont besoin d'être exécutées à plusieurs reprises, pendant le déroulement d'un script.

La définition d'une fonction doit être faite avant son premier appel.

Syntaxe :

```
function mafonction {  
Commande_1  
Commande_2  
...  
}
```

Appel de fonction avec : `mafonction`

Une fonction peut être appelée aussi bien à partir du programme principal, qu'à partir d'une autre fonction.

Dans un script contenant des fonctions, les commandes situées en dehors des corps de fonctions sont **exécutées séquentiellement**.

Dès qu'une fonction est définie, elle est considérée par le Shell comme une commande interne.

Par défaut, le code de retour d'une fonction correspond au code de la dernière commande exécutée dans la fonction.

Pour forcer un **code de retour global** à la fonction, il faut utiliser la commande : `return`

Il est possible de passer des arguments à une fonction : `mafonction 1 2 3`

A l'intérieur du corps de la fonction, les valeurs transmises seront accessibles via les **variables positionnelles** habituelles : **\$1, \$2 et \$3** indépendamment des variables de position du programme principal, et d'éventuelles autres fonctions.

D) Substitutions

1. Substitution de commande

Le **mécanisme** habituel : `var=`pwd``

Peut être écrit de la façon suivante : `var=$(pwd)`

2. Substitution de variable

Ce mécanisme permet, entre autres, d'attribuer une valeur par défaut aux variables.

<code>echo \${variable:-valeur}</code>	Si la variable est vide, echo affichera valeur
<code>echo \${variable:=valeur}</code>	Si la variable est vide, valeur est affecté à variable, avant l'affichage
<code>echo \${variable:+valeur}</code>	Si la variable n'est pas vide, l'expression est substituée par valeur
<code>echo \${variable:?message}</code>	Si la variable est vide, affichage du nom de variable et du message, puis arrêt du script

3. Substitution de processus

La substitution de commande est classique. Deux syntaxes sont disponibles :

\$(cmd)
ou : `cmd`

Exemple : `grep 'body' $(find / -name '*.html')`
 Recherche de la chaîne de caractères "body", sur tout le disque, dans les fichiers dont le nom se termine par ".html".

La substitution de processus (*process substitution*, en anglais) est identique, sauf que le résultat de la commande est considéré comme un fichier, et non comme une chaîne de caractères.

Syntaxe : <(cmd)

Le résultat du processus doit donc être transmis à une commande, qui attend un fichier en argument.

Exemple : `cat <(ls -R /)`
 Affichage du contenu de tous les fichiers du disque.

E) Types de données

Le *Bash*, ainsi que le *Ksh*, permettent de déclarer des variables de type entier (pour accélérer les calculs), avec la commande `typeset`

Exemple : `typeset -i chiffre
chiffre=3*9
echo $chiffre`

27

F) Longueur de chaîne

Le symbole # permettra d'obtenir la taille d'une variable, en nombre d'octets.

Exemple : `auto=Verrue
echo "La taille de $auto est ${#auto}"
La taille de Verrue est 6`

G) Traitement des chaînes

4. Extraction d'une sous-chaîne

`${chaine:position}`

Extrait une sous-chaîne de `$chaine` à partir de la position `$position`.

```
chaine=ABCDEFGHIJ
echo ${chaine:7} # HIJ
```

La valeur `$position` part de zéro.

Les opérateurs `#`, `##`, `%` et `%%`

Ils permettent d'extraire des portions de chaînes de caractères, en association avec les métacaractères utilisés sur les fichiers.

<code>#</code>	Cherche la plus <u>petite occurrence</u> en début de texte
<code>##</code>	Cherche la plus <u>grande occurrence</u> en début de texte
<code>%</code>	Cherche la plus <u>petite occurrence</u> en fin de texte
<code>%%</code>	Cherche la plus <u>grande occurrence</u> en fin de texte

Le résultat sera le **complément de la chaîne trouvée**.

Exemple :	<code>var=12ABC12ABC</code>		
	<code>echo \${var#1*B}</code>	affichera	<code>C12ABC</code>
	<code>echo \${var##1*B}</code>	affichera	<code>C</code>
	<code>echo \${var%*B*C}</code>	affichera	<code>12ABC12A</code>
	<code>echo \${var%*B*C}</code>	affichera	<code>12A</code>

Remarque : Le premier ou le dernier caractère doivent correspondre.

`expr substr $chaine $position $longueur`

Extrait `$longueur` caractères à partir de `$chaine` en commençant à `$position`.

```
chaine=ABCDEFGHIJ
echo `expr substr $chaine 7 2` # GH
```

La valeur `$position` part de un.

5. Remplacement de sous-chaîne

`${chaine/souschaine/remplacement}`

Remplace la première correspondance de `souschaine` par `remplacement`.

`${chaine//souschaine/remplacement}`

Remplace toutes les correspondances de `souschaine` avec `remplacement`.

```
chaine=abcabcabc12
echo ${chaine/abc/xyz} # xyzabcabc12
echo ${chaine//abc/xyz} # xyzxyzxyz12
```

`${chaine/#souschaine/remplacement}`

Si *souschaine* correspond au *début* de *chaine*, substitue *remplacement* à *souschaine*.

`${chaine/%souschaine/remplacement}`

Si *souschaine* correspond à la *fin* de *chaine*, substitue *remplacement* à *souschaine*.

H) Opérateur ..

```
echo {1..5}
1 2 3 4 5
```

```
echo {a..f}
a b c d e f
```

I) Tableaux

La construction de "Tableaux de variables", parfois appelés **Champs**, s'effectue ainsi :

```
legume[0]=Tomate
legume[1]=Haricot
legume[2]=Fayot
```

ou avec :

```
declare -a legume
legume=(Tomate Haricot Fayot)
```

Dans les 2 cas, les possibilités d'utilisation seront identiques :

```
i=1
echo ${legume[i]}          (départ à zéro - second légume, donc ...)
Haricot
echo ${#legume[*]}         (Nombre de légumes ...)
3
```

J) Commandes

let Remplace (et cohabite avec) la commande *expr*

Exemple : `let "chiffre=chiffre+3"`
`echo $chiffre`
`3`

select Permet de créer des menus simples.

Syntaxe : `select variable in liste_de_valeurs`
`do`
 `Liste_de_commandes`
`done`

Exemple :

```

PS3="Numéro : "
select nom in Jean Paul Jacques
do
    if [ -n "$nom" ]
    then
        echo "C'est $nom qui est retenu"
    fi
done

```

A l'exécution, les éléments suivants seront proposés :

- 1) Jean
 - 2) Paul
 - 3) Jacques
- Numéro :

Remarque : Par défaut, *PS3* contient "#?"

Si un **numéro de la liste** est saisi, après le *prompt prévu*, la **variable nom** contient le *texte correspondant au nombre tapé*.

Si l'*élément saisi* ne correspond à aucune des valeurs de la liste, la valeur saisie est placée dans la variable **REPLY**

Un **Ctrl-D** permet d'interrompre le traitement du "select".

Autrement, la sortie du "do ... done" s'effectue avec une instruction "break".

K) Variables

Le *Shell* exploite quelques variables particulières; En voici quelques unes :

HISTFILE	Nom du fichier historique
HISTSIZE	Nombre de commandes du fichier historique (par défaut : 1 000)
PWD	Chemin d'accès du répertoire courant
RANDOM	Nombre aléatoire, compris entre 0 et 32767
REPLY	Contient la saisie de l'utilisateur après une commande <i>read</i> , si <u>aucun nom de variable n'a été spécifié</u>
SECONDS	Temps écoulé (en secondes !) depuis l'appel du shell actif