

# Shell

## Sécurisation exécution

### set

set -o pipefail	Permet de retourner une erreur quand la première partie d'un pipe est en erreur
set -o errexit (ou set -e)	Force l'arrêt du script en cas d'erreur d'une commande
set -o nounset (ou set -u)	Sort en erreur si une variable est utilisée sans avoir été déclarée préalablement
set -o errtrace (ou set -E)	
set -x	Mode debug, chaque ligne est affichée lors de l'exécution

## Variables

### Paramètres

\$0	Nom de la commande en cours d'exécution (ou shell courant)
\$1..\$9	Arguments de la ligne de commande ou de la procédure en cours
\$#	Nombre d'arguments
\$*	Liste des arguments (sauf \$0) en 1 seul argument en utilisant le séparateur \$IFS (“\$1 \$2 \$3 ...\$n”)
\$@	Liste des arguments (sauf \$0) en n arguments (“\$1” “\$2” “\$3” ...“\$n”)
Shift	Décale les arguments à gauche, \$0 inchangé, \$1 perdu, \$2 passe dans \$1...

### Entrées/Sorties

- 0 : entrée standard
- 1 : sortie standard
- 2 : sortie d'erreurs standard

### Exemples

```
1>/dev/null : redirige <stdout> vers la poubelle
2>/dev/null : redirige les erreurs vers la poubelle
1>&2 : redirige <stdout> vers la sortie d'erreurs
```

### Variables prédéfinies

\$?	Code de retour dernière commande (0 à 255). Code nul = vrai, sinon faux
\$\$	Numéro de processus en cours d'exécution

\$!	Numéro du dernier processus lancé en tâche de fond
\$_	paramètre le plus récent (ou le chemin abs de la commande pour démarrer le shell courant immédiatement après le démarrage)
\$-	options actuelles définies pour le shell
\$FUNCNAME	Nom de la fonction
\$LINENO	Numéro de ligne
\$IFS	Internal Field Separator (utile pour les boucles : IFS=\$'\n')
~dp0	Répertoire où se situe le script en cours d'exécution
%CD%	Répertoire courant

[Index des variables](#)

[Paramètres spéciaux](#)

[Variables shell](#)

## Bash

PS1	Prompt par défaut de la ligne de commande
PS2	Prompt pour les commandes sur plusieurs lignes suite à \. Par défaut affiche "> " sur les lignes suivantes
PS3	Prompt pour les boucles select
PS4	Prompt pour l'exécution en mode debug avec "set -x"
PROMPT_COMMAND	Commande exécuté juste avant l'affichage de la variable PS1

[https://www.thegeekstuff.com/2008/09/bash-shell-take-control-of-ps1-ps2-ps3-ps4-and-prompt\\_command/](https://www.thegeekstuff.com/2008/09/bash-shell-take-control-of-ps1-ps2-ps3-ps4-and-prompt_command/)

## Alias

\!*	désigne la liste des paramètres
\!^	désigne le premier paramètre
\!\$	désigne le dernier paramètre
\!:n	désigne le nième paramètre

## Tableaux

```
tableau[0]="truc"
tableau[1]="machin"
autre_tableau=("truc" "machin")
```

```
`${#tableau[@]}` indice du dernier élément du tableau (nombre éléments -1)
`${!fruits[@]}` donne la liste des indices du tableau (pour utilisation dans une boucle par exemple)
```

```
echo ${tableau[1]}
echo ${tableau[*]} # affiche tous les éléments
echo ${tableau[@]} # autre forme, même résultat
```

## Calcul

Incrémenter de 1 la variable \$var:

```
TOTO=`expr $var + 1`  
let "var=var+1"  
((var=var+1))  
((var++))
```

## Formatage

```
printf "%*s  %*s  %*s\n", $lng1, $var1, $lng2, $var2, $lng3, $var3
```

## Shell parameter expansion

[https://www.gnu.org/software/bash/manual/html\\_node/Shell-Parameter-Expansion.html](https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html)

---

## Manipulation de chaînes

### Majuscule/Minuscule

```
texte_premiere_maj="${texte^}" # passe la première lettre en majuscule  
texte_maj="${texte^^}" # passe la chaîne complète en majuscule  
texte_premiere_min="${texte,}" # passe la première lettre en minuscule  
texte_min="${texte,,}" # passe la chaîne complète en minuscule
```

### Remplacement de caractère

**`${variable//pattern/replacement}`**

```
# remplace ; par ,  
modif_variable=${variable//;/,}  
# enlever toutes les " dans une chaîne  
modif_variable=${variable//\"/}
```

### Extraire une sous chaîne

### Exemples

```
var=banane
```

%a*	plus petite chaîne commençant par a (ex : <code>\${var%a*}=ban</code> )
%%a*	plus grande chaîne commençant par a
#*a	plus petite chaîne terminant par a (ex : <code>\${var#*a}=nane</code> )
##*a	plus grande chaîne terminant par a (ex : <code>\${var##*a}=ne</code> )

## Utilisation d'un séparateur

```
var="param1:param2"  
IFS=:  
set var  
# var devient param1 param2  
param1=$1  
param2=$2
```

## Expressions régulières

Les expressions régulières peuvent être utilisées avec bash depuis la version 3 en utilisant l'opérateur =~

## Syntaxe

```
if [[ $VALEUR =~ regex ]]; then  
    echo "OK"  
else  
    echo "KO"  
fi
```

## Capture

Les variables sont `${BASH_REMATCH[1]}` à `${BASH_REMATCH[n]}` au lieu de `$1` à `$n`

## Vérification expression régulière

<https://regex101.com/>

## Source

[https://fr.wikibooks.org/wiki/Programmation\\_Bash/Regex](https://fr.wikibooks.org/wiki/Programmation_Bash/Regex)

## Lire un fichier csv

```
# définir le séparateur , ou ;
IFS=,
while read -r colonne1 colonne2 colonne3; do
    echo $colonne1
    echo $colonne2
    echo $colonne3
done < $fichier_csv
```

## Construire une commande dans une variable

Pour construire une commande à exécuter dans une variable il faut utiliser un tableau pour éviter des problèmes avec les ' ou ".

```
# ne pas faire
USER="toto"
IP="192.168.0.1"
OPT="-o StrictHostKeyChecking=no"
CMD="$OPT $USER@$IP"
ssh $CMD

# utiliser un tableau à la place
CMD=(-o StrictHostKeyChecking=no toto@192.168.0.1)
ssh ${CMD[@]}
```

---

## Tests

### Opérateurs

!	négation d'une expression
-a	et logique
-o	ou logique (le et logique a une préséance plus grande)
\(expr\)	parenthèses pour regrouper les expressions (\ pour éviter interprétation)

### Fichiers

-r	vrai si le fichier existe et accessible en lecture pour l'utilisateur
-w	vrai si le fichier existe et accessible en écriture pour l'utilisateur
-x	vrai si le fichier existe et accessible en exécution pour l'utilisateur
-f	vrai si le fichier existe et est un fichier normal
-d	vrai si le fichier existe et est un répertoire
-c	vrai si le fichier existe et est du type spécial caractère
-b	vrai si le fichier existe et est du type spécial bloc
-s	vrai si le fichier existe et a une taille non nulle

-L	vrai si le fichier est un lien
-e	vrai si fichier régulier ou lien

## Chaînes de caractères

-z s1	vrai si la chaîne s1 est de longueur nulle
-n s1	vrai si la chaîne s1 contient au moins un caractère
s1=s2	vrai si les 2 chaînes sont égales (attention aux blancs encadrant le signe =)
s1!=s2	vrai si les 2 chaînes sont différentes
str	Vérifie si str n'est pas la chaîne vide; s'il est vide, il renvoie false

## Entiers

-ne différent	-eq égal
-gt supérieur	-lt inférieur
-ge supérieur ou égal	-le inférieur ou égal

## Fonctions

### Fichiers

basename	extraie le nom du fichier après le dernier /
dirname	extraie le chemin d'accès avant le dernier /
readlink <lien>	obtient la cible du lien
readlink -f <lien>	obtient la cible du lien avec le chemin complet

### Mathématique

+	Addition `expr a + b`
-	Soustraction `expr a - b`
*	Multiplication `expr a * b`
/	Division `expr a / b`
%	Modulo. Divise l'opérande de gauche par l'opérande de droite et renvoie le reste. `expr a % b`

### Saisie clavier : read

#### options

- -p : affiche un prompt
- -s : mode silencieux (n'affiche pas ce qui est tapé)
- -t : définit un délai d'expiration
- -a : stocke les entrées dans un tableau
- -n : limite le nombre de caractères à lire
- -r : mode raw (n'interprète pas les caractères d'échappement)

## exemples

```
# lecture simple
echo "Entrez votre nom : "
read nom
echo $nom

# lecture avec prompt intégré
read -p "Entrez votre âge : " age
echo "$age"

# lecture silencieuse pour mot de passe
read -sp "Entrez votre mot de passe : " mdp

# Lecture avec délai d'expiration (en secondes)
read -t 5 -p "Vous avez 5 secondes pour répondre : " reponse

# Lecture de plusieurs variables
read -p "Entrez prénom et nom : " prenom nom
echo "Prénom : $prenom, Nom : $nom"

# Lecture d'un tableau
read -a tableau -p "Entrez plusieurs mots : "
echo "Premier mot : ${tableau[0]}"
echo "Deuxième mot : ${tableau[1]}"
```

---

## Structures de contrôle

### Prise de décision

#### if

```
if [ expression ]
then
    Statement
else
    Statement
fi
```

```
if [ expression 1 ]
then
    Statement
elif [ expression 2 ]
then
    Statement
elif [ expression 3 ]
```

```
then
  Statement
else
  Statement
fi
```

## case

```
option="${1}"
case ${option} in
  -f) FILE="${2}"
      echo "File name is $FILE"
      ;;
  -d) DIR="${2}"
      echo "Dir name is $DIR"
      ;;
  *)
      echo "`basename ${0}`:usage: [-f file] | [-d directory]"
      exit 1 # Command to come out of the program with status 1
      ;;
esac
```

## Boucles

### Boucle while

```
a=0
while [ $a -lt 10 ]
do
  echo $a
  a=`expr $a + 1`
done
```

### Boucle for

```
for var in 0 1 2 3 4 5 6 7 8 9
do
  echo $var
done
```

```
for i in $(seq 0 9); do echo $i; done
```

```
fruits=("pomme" "orange" "banane" "fraise" "kiwi")
```

```
for fruit in "${fruits[@]}"; do
  echo "Fruit: $fruit"
```

```
done
for (( i=0; i<${#fruits[@]}; i++ )); do
    echo "Fruit $i: ${fruits[$i]}"
done
for index in "${!fruits[@]}; do
    echo "Fruit $index: ${fruits[$index]}"
done
```

## Boucle until

```
a=0
until [ ! $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```

## Boucle sélection

Permet de créer un menu numéroté

```
PS3="Please make a selection => " # sans espace autour du signe =
select DRINK in tea cofee water juice appe all none
do
    case $DRINK in
        tea|cofee|water|all)
            echo "Go to canteen"
            ;;
        juice|appe)
            echo "Available at home"
            ;;
        none)
            break
            ;;
        *) echo "ERROR: Invalid selection"
            ;;
    esac
done
```

## Contrôle de boucle

break	sort de l'exécution de la boucle. Possibilité d'indiquer un nombre pour sortir de plusieurs niveau de boucle
continue	passse à l'itération suivante de la boucle . Possibilité également de préciser un nombre pour indiquer un niveau supérieur de boucle

## Procédures/Fonctions

```
function maFonction()  
{  
    ...  
}  
maFonction()  
{  
    ...  
}
```

Les deux syntaxes ont leur avantage :

- L'absence du mot-clé permet au script d'être compatible avec les shells Bourne et Korn.
- La présence du mot-clé permet d'éviter une collision de nom avec les alias.

---

## Commandes

### Nom de fichier

#### **basename**

Extraction du nom de fichier (ou repertoire) du chemin donné en entrée. En fait donne la chaîne après le dernier /.

#### **dirname**

Extraction de la partie repertoire du chemin donné en entrée

---

## Astuces

### Créer un fichier sans éditeur

```
cat << tagFIN > nomdufichier  
....  
...  
tagFIN
```

## Sources

- [Programmation batch](#)
- [Manipulation chaîne de caractères](#)

From:

<https://wiki.iot-acis.fr/> - **Wiki**

Permanent link:

<https://wiki.iot-acis.fr/doku.php?id=all:bibles:langages:shell>

Last update: **2026/03/30 14:13**

