

Module : Système et Scripting

Année Universitaire
2024-2025

Planification & Informations

- **Chapitre 1** : Introduction & Commandes de base
- **Chapitre 2** : Langage de programmation Shell
- **Chapitre 3** : Conditions & Boucles en Shell
- **Chapitre 4** : Sous-programmes en Shell

Système & Scripting

Chapitre II : Langage de programmation Shell

Plan

- 1) Shell : Le principe des scripts
- 2) Fonctionnement des scripts
- 3) Exécution des scripts
- 4) composants des scripts
- 5) Déclaration des variables
- 6) Affichage des variables avec echo
- 7) Saisie des variables avec read
- 8) Les variables d'environnement
- 9) Les variables spéciales
- 10) Tableaux
- 11) Expr : arithmétique
- 12) Expr : manipulation des chaînes de caractères

Le principe des scripts

Un script est un ensemble de commandes qui sont regroupées dans un fichier exécutable. Les avantages sont :

- Assembler des commandes simples pour réaliser des actions complexes
- Regrouper des actions que l'on désire exécuter plus d'une fois
- Réaliser un outil à mettre à la disposition de plusieurs personnes
- Un script est un programme interprété et non compilé.

L'intérêt de maîtriser l'art du scripting réside dans la capacité d'automatiser des tâches complexes ainsi que d'être capable de lire et de modifier les scripts existants sur un système.

Fonctionnement des scripts

Pour fonctionner un script doit répondre à plusieurs critères :

- 1) Etre exécutable pour l'utilisateur au niveau de la protection du fichier

chmod +x nom_du_script.sh

- 2) Avoir l'appel à l'interpréteur dans sa première ligne

#!/bin/bash

Fonctionnement des scripts

- La première ligne commençant par `#!` indique à quel interpréteur le script doit être confié.
- Il existe de nombreux interpréteurs possibles, par exemple:
 - `#!/bin/sh`
 - `#!/usr/bin/tcl`
 - `#!/bin/bash`
 - `#!/bin/sed -f`
 - `#!/usr/bin/perl`
 - `#!/bin/awk -f`

Exécution des scripts

1) Exécution via un Shell spécifique

Si le script est exécutable, il suffit de passer le script en argument à un Shell.

bash monscript.sh

2) Exécution directe avec les permissions appropriées

Si le script est configuré avec des droits d'exécution (chmod +x), il peut être lancé directement en indiquant son chemin relatif ou absolu. Par exemple :

./monscript.sh

3) Exécution dans le contexte du Shell courant

Une autre méthode consiste à exécuter le script dans le même environnement que le shell appelant, ce qui signifie que les modifications apportées (par exemple, l'exportation de variables) affecteront le Shell actuel.

. ./monscript.sh ou

source monscript.sh

Composants des scripts

- ❖ une commande Shell de base : find, cut ou echo
- ❖ une variable : \$var
- ❖ un commentaire : #ceci est un commentaire: Ce qui suit ne sera pas exécuté. On peut commencer un commentaire n'importe où sur une ligne et il se termine obligatoirement à la fin de la ligne. Par contre, les # dans des chaînes de caractères ne sont pas des commentaires.
- ❖ une fonction : définition d'un sous programme ou procédure
- ❖ une structure de contrôle : if-then-else, for ou while
- ❖ # est un commentaire, ; est un séparateur de commandes, il permet de placer plusieurs commandes sur une seule ligne.

```
echo "Le fichier $nomfichier existe"; cp $nomfichier $nomfichier.sauve
```

Déclaration des variables

- ❖ création un nouveau script variables.sh :

```
$ nano variables.sh
```

- ❖ La première ligne scripts doit indiquer quel shell est utilisé .

```
#!/bin/bash
```

- ❖ Définition des variables.

- Toute variable possède un nom et une valeur :

```
#!/bin/bash  
message='Bonjour ESPRIT '
```

- ❖ Exécution du script

```
$ ./variables.sh
```

Affichage des variables avec echo

- La commande echo permet d'afficher une ligne.

```
$ echo Salut ESPRIT
Salut ESPRIT
```

- lors de insertion des retours à la ligne, il faudra activer l'option -e et utiliser le symbole \n :

```
$ echo -e "Message\n Autre ligne"
Message
Autre ligne
```

- Afficher une variable

```
#!/bin/bash
message='Bonjour ESPRIT'
echo $message
```

Affichage des variables avec echo

❖ Affichage du texte et variable.

```
#!/bin/bash  
message='Bonjour Esprit'  
echo 'Le message est : $message'
```



Le problème est que cela ne fonctionne pas car cela affiche :

```
Le message est : $message
```

❖ La compréhension des quotes

- Il existe trois types de quotes :
 - les apostrophes ' ' (simples quotes)
 - les guillemets " " (doubles quotes)
 - les accents graves ` ` (back quotes)

Affichage des variables avec echo

Les « quotes »

Avec de simples quotes: la variable n'est pas analysée et le \$ est affiché tel quel:

```
#!/bin/bash
message='Bonjour Esprit'
echo 'Le message est : $message'
```

```
Le message est : $message
```

Avec les doubles quotes: ils demandent à bash d'analyser le contenu du message s'il a trouvé des symboles spéciaux (comme des variables), il les interprète.

```
#!/bin/bash
message='Bonjour Esprit'
echo "Le message est : $message"
```

```
Le message est : Bonjour Esprit
```

Avec les back quotes: ils demandent à bash d'exécuter ce qui se trouve à l'intérieur.

```
#!/bin/bash
message=`pwd`
echo "Vous êtes dans $message"
```

```
Vous êtes dans /home/Esprit
```

Saisie des variables avec read

- La commande read lit son entrée standard et affecte les valeurs saisies dans la ou les variables passées en argument.

```
#!/bin/bash
read nom
echo "Bonjour $nom "
```

- Affecter simultanément une valeur à plusieurs variables

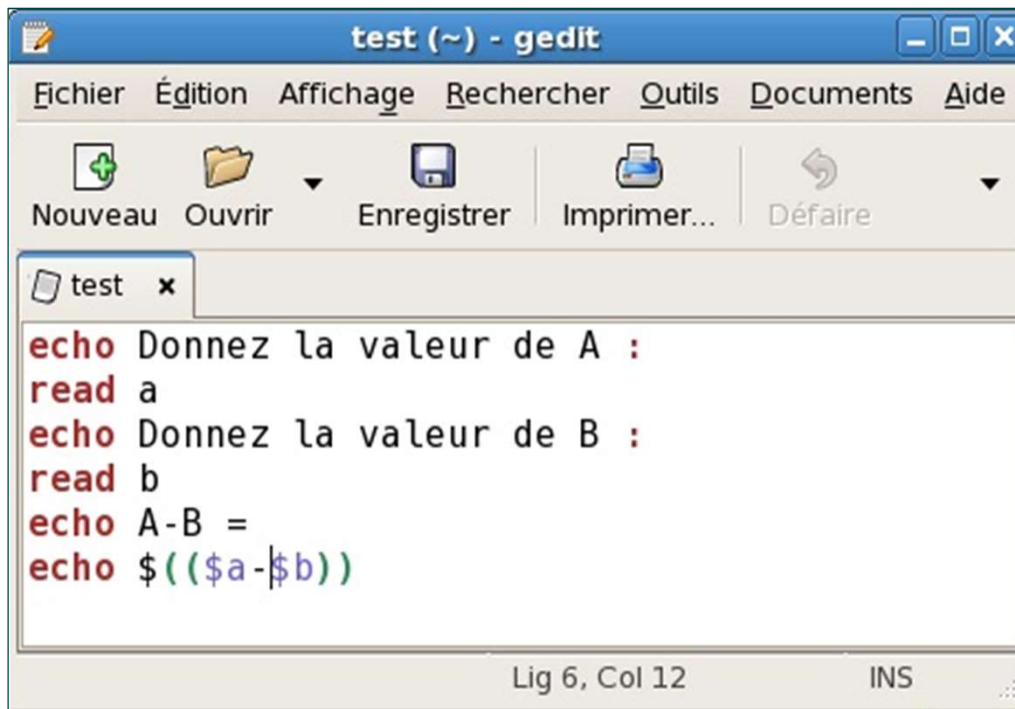
```
#!/bin/bash
read nom prenom
echo "Bonjour $nom $prenom"
```

- Afficher un message de prompt

```
#!/bin/bash
read -p 'Entrez votre nom : ' nom
echo "Bonjour $nom "
```

Saisie des variables avec read

Exemple



```
test (~) - gedit
Fichier Édition Affichage Rechercher Outils Documents Aide
Nouveau Ouvrir Enregistrer Imprimer... Défaire
test x
echo Donnez la valeur de A :
read a
echo Donnez la valeur de B :
read b
echo A-B =
echo $((a-b))
Lig 6, Col 12 INS
```



```
root@localhost:~
Fichier Édition Affichage Terminal Onglets Aid
[root@localhost ~]# ./test
Donnez la valeur de A :
10
Donnez la valeur de B :
4
A-B =
6
[root@localhost ~]#
```

Les variables d'environnement

- une variable définie dans un programme A ne sera pas utilisable dans un programme B.
- Les variables d'environnement sont des variables que l'on peut utiliser dans n'importe quel programme. On parle aussi parfois de variables globales.

➤ la commande env :

```
$ printenv
PATH=:/usr/share/glade3/pixmaps
TERM=xterm
SHELL=/bin/bash
USER=ESPRIT
PATH=/home/esprit/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin: PWD=/home/mateo21/bin
EDITOR=nano
HOME=/home/esprit
```

Les variables d'environnement

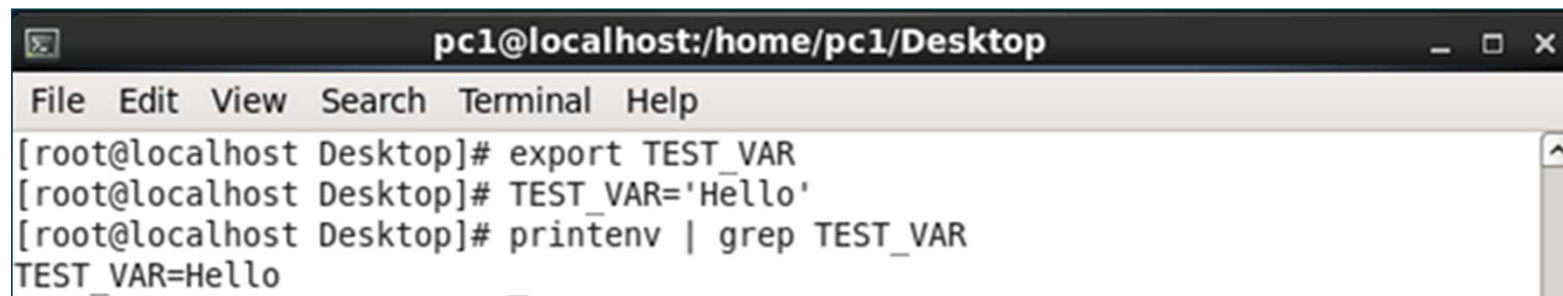
- **SHELL**: indique quel type de shell est en cours d'utilisation (sh, bash, ksh)
- **PATH**: une liste des répertoires qui contiennent des exécutables que vous souhaitez pouvoir lancer sans indiquer leur répertoire.
- **EDITOR**: l'éditeur de texte par défaut qui s'ouvre lorsque cela est nécessaire ;
- **HOME**: la position de votre dossierhome ;
- **PWD**: le dossier dans lequel vous vous trouvez ;

```
#!/bin/bash  
echo "Votre éditeur par défaut est $EDITOR"
```

```
Votre éditeur par défaut est nano
```

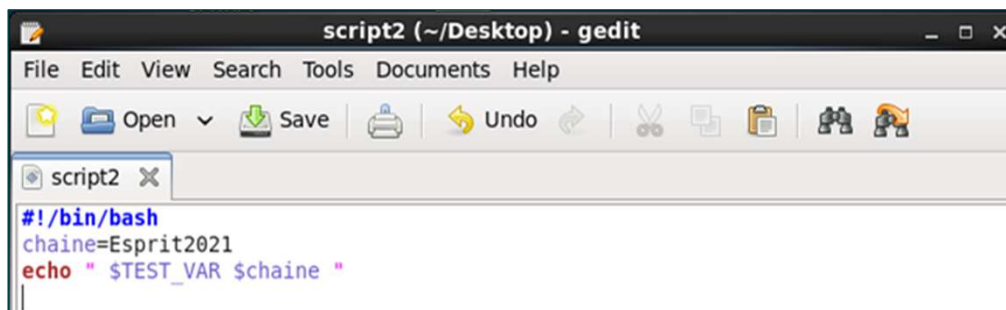
Les variables d'environnement

Création des variables d'environnement

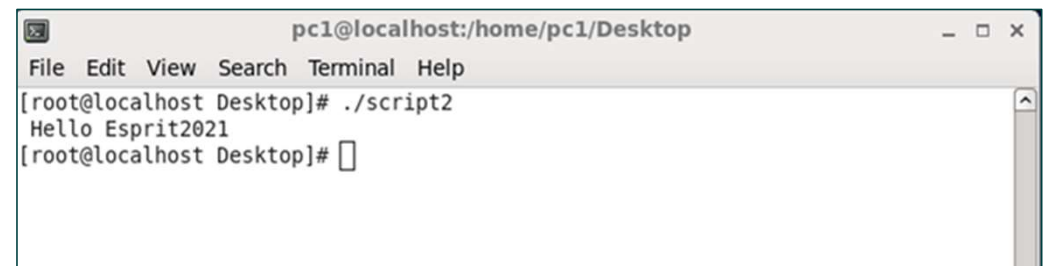


```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# export TEST_VAR
[root@localhost Desktop]# TEST_VAR='Hello'
[root@localhost Desktop]# printenv | grep TEST_VAR
TEST_VAR=Hello
```

Utilisation des variables d'environnement



```
script2 (~/Desktop) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
script2
#!/bin/bash
chaine=Esprit2021
echo " $TEST_VAR $chaine "
```

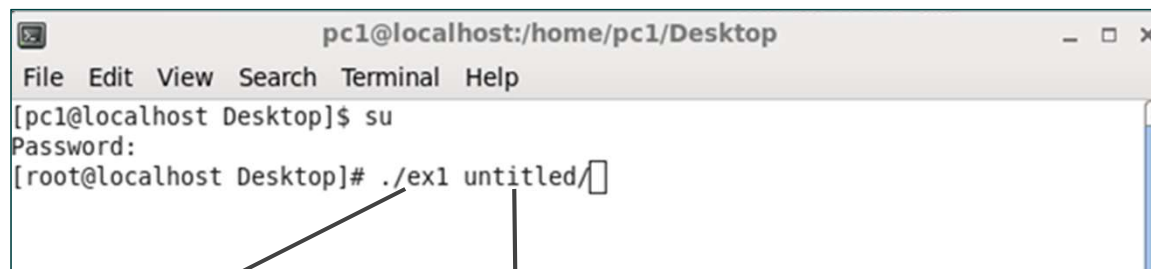


```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# ./script2
Hello Esprit2021
[root@localhost Desktop]#
```

Les variables spéciales

Le script Shell récupère les arguments de la ligne de commande dans des variables réservées appelées Paramètres Positionnels

<i>Variables spéciales</i>	<i>Signification</i>
<code>##</code>	Nombre d'arguments reçus par le script
<code>\$0</code>	Le nom du script lui-même
<code>\$1 \$2 ... \$9 \${10}</code>	<code>\$1</code> est la valeur du premier argument, <code>\$2</code> la valeur du second...
<code>\$*</code>	Liste des arguments

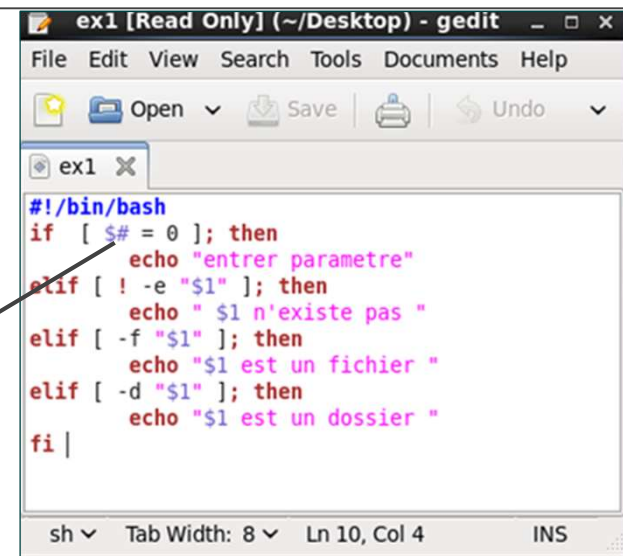


```
pcl@localhost:/home/pcl/Desktop
File Edit View Search Terminal Help
[pcl@localhost Desktop]$ su
Password:
[root@localhost Desktop]# ./ex1 untitled/
```

`$0`

Le 1^{er} argument `$1`

Le Nombre d'arguments



```
ex1 [Read Only] (~/Desktop) - gedit
File Edit View Search Tools Documents Help
ex1
#!/bin/bash
if [ $# = 0 ]; then
    echo "entrer parametre"
elif [ ! -e "$1" ]; then
    echo " $1 n'existe pas "
elif [ -f "$1" ]; then
    echo "$1 est un fichier "
elif [ -d "$1" ]; then
    echo "$1 est un dossier "
fi
```

Substitution des variables

- **\$ Substitution de variable (contenu d'une variable).**

```
var1=5  
var2=23Esprit
```

```
echo $var1  
# 5  
echo $var2  
# 23Esprit
```

- **\${} substitution de paramètres**
- \${parametre} Identique à \$parametre, c'est-à-dire la valeur de la variable paramètre.
- **Concaténation de variables avec des chaînes**

Peut être utilisé pour concaténer des variables avec des suites de caractères (strings).

```
votre_id=${USER}-sur-${HOSTNAME}
```

Tableaux

1. Création d'un tableau

- Quatre méthodes pour créer un tableau:

- Avec la commande declare et l'option -a :

```
declare -a nom-tableau=(valeur0 valeur1 valeur2 ...)
```

- Directement :

```
nom-tableau=(valeur0 valeur1 ...)
```

- Autre syntaxe :

```
nom-tableau=([indice0]=valeur0 [indice1]=valeur1 ...)
```

- Assigner un élément

```
nomtableau[indice]=valeur
```

```
$ tab[0]=10 $ tab[2]=12
```

2. Affichage des éléments

Exemple: Affichage de les éléments d'indice 0 et d'indice 2:

```
$ echo ${tab[0]}  
10
```

```
$ echo ${tab[2]}  
12
```

Tableaux

3. Accès à toutes les valeurs d'un tableau

- Tous les éléments d'un tableau sont accessibles avec chacune de ces deux syntaxes :

```
${nom-tableau[*]}
```

```
${nom-tableau[@]}
```

4. Comptage des éléments dans un tableau

- Le nombre d'éléments d'un tableau est accessible par chacune de ces deux syntaxes :

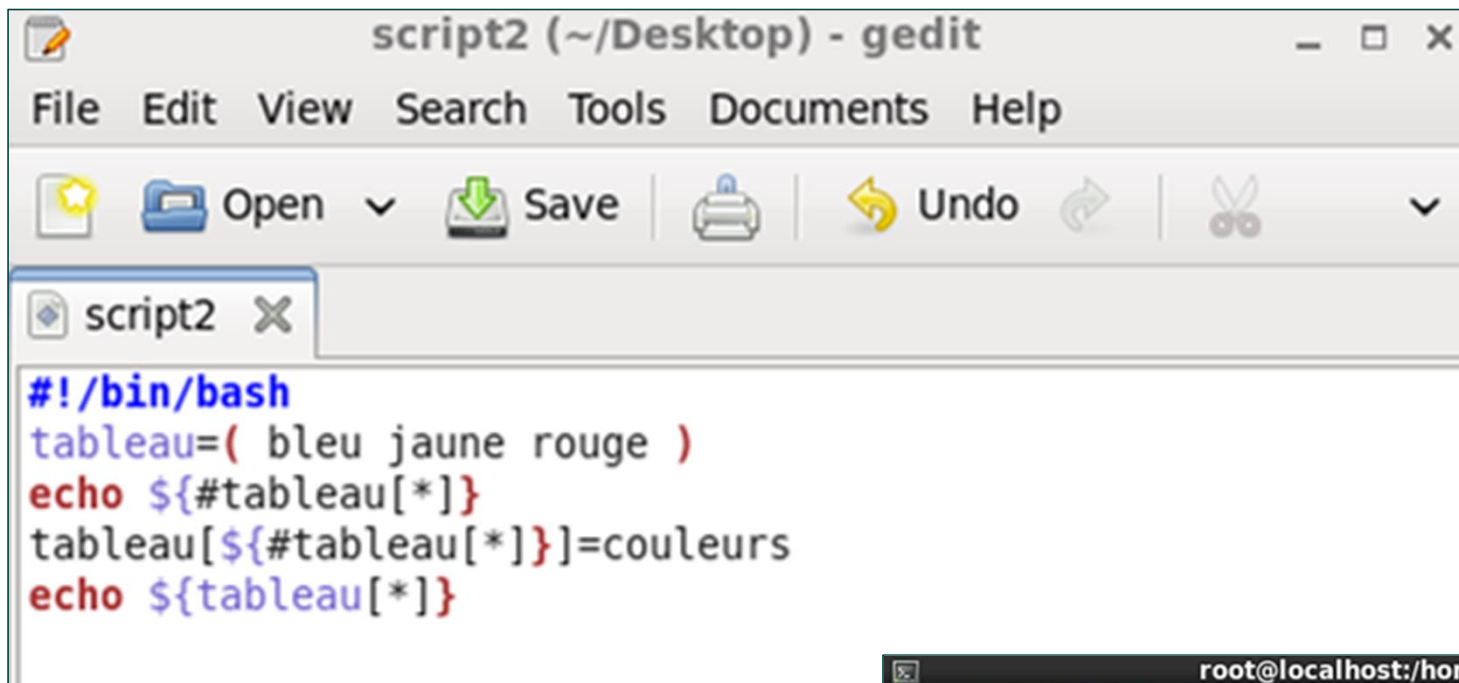
```
${#nomtableau[*]}
```

```
${#nomtableau[@]}
```

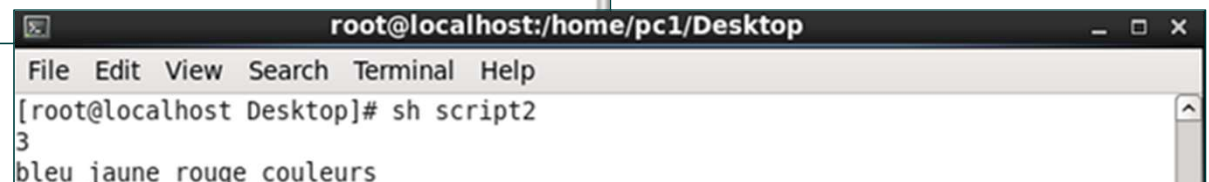
Tableaux

5- Ajout d'un élément à la fin du tableau:

```
tableau[${#tableau[*]}]=élément
```



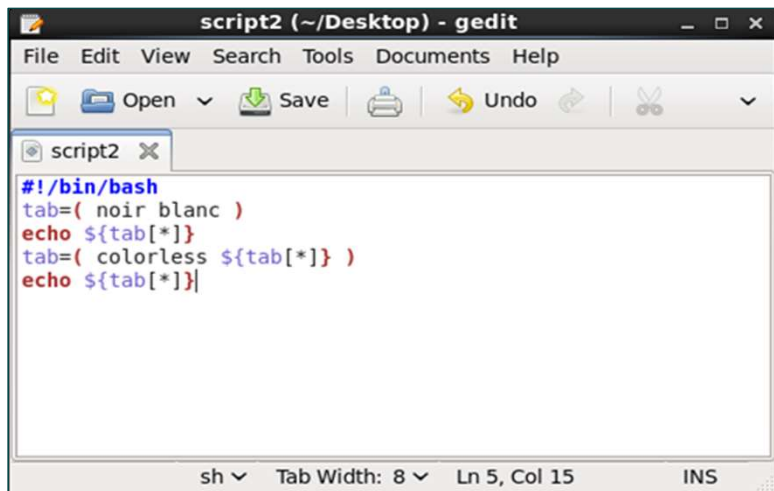
```
#!/bin/bash  
tableau=( bleu jaune rouge )  
echo ${#tableau[*]}  
tableau[${#tableau[*]}]=couleurs  
echo ${tableau[*]}
```



```
root@localhost:/home/pc1/Desktop  
[root@localhost Desktop]# sh script2  
3  
bleu jaune rouge couleurs
```

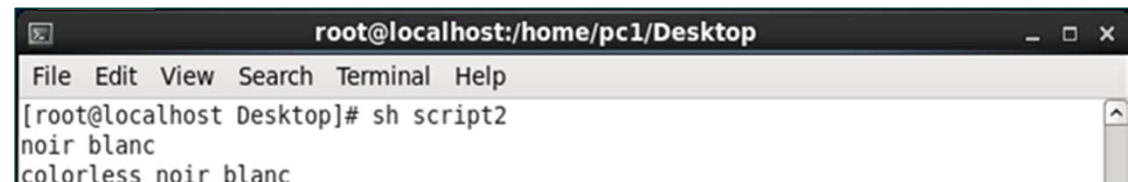
Tableaux

6- Ajout d'un élément au début d'un tableau :



```
#!/bin/bash
tab=( noir blanc )
echo ${tab[*]}
tab=( colorless ${tab[*]} )
echo ${tab[*]}
```

```
tableau=( element ${tableau[*]} )
```



```
root@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
noir blanc
colorless noir blanc
```

7- Supprimer un tableau:

```
unset nom-tableau
```

8- Supprimer la case d'un tableau :

```
unset nom-tableau[indice]
```

Expr : arithmétique

La commande « expr » évalue ses arguments et écrit le résultat sur la sortie standard. La première utilisation de « expr » concerne les opérations arithmétiques simples.

Les opérateurs arithmétiques sont :

- + : addition ;
- - : soustraction ;
- * : multiplication ;
- / : division entière ;
- % : reste de la division ;
- \(et \) : parenthèses.

Expr : arithmétique

Exemple: Syntaxe1

```
#!/bin/bash
a=10
b=3
# Addition
result=`expr $a + $b`
echo "Addition: $result" # Affiche Addition: 13
# Soustraction
result=`expr $a - $b`
echo "Soustraction: $result" # Affiche Soustraction: 7
# Multiplication
result=`expr $a \* $b`
echo "Multiplication: $result" # Affiche Multiplication: 30

# Division entière
result=`expr $a / $b`
echo "Division: $result" # Affiche Division: 3
# Reste de la division
result=`expr $a % $b`
echo "Reste: $result" # Affiche Reste: 1
# Parenthèses
result=`expr \( $a + $b \) \* 2`
echo "Parenthèses: $result" # Affiche Parenthèses: 26
```

```
esprit@esprit:~$ ./script3
Addition: 13
Soustraction: 7
Multiplication: 30
Division: 3
Reste: 1
Parenthèses: 26
```

Expr : arithmétique

Exemple Syntaxe2

```
#!/bin/bash
a=10
b=3
# Addition
result=$(expr $a + $b)
echo "Addition: $result" # Affiche Addition: 13
# Soustraction
result=$(expr $a - $b)
echo "Soustraction: $result" # Affiche Soustraction: 7
# Multiplication
result=$(expr $a \* $b)
echo "Multiplication: $result" # Affiche Multiplication: 30
# Division entière
result=$(expr $a / $b)
echo "Division: $result" # Affiche Division: 3
# Reste de la division
result=$(expr $a % $b)
echo "Reste: $result" # Affiche Reste: 1
# Parenthèses
result=$(expr \( $a + $b \) \* 2)
echo "Parenthèses: $result" # Affiche Parenthèses: 26
```

```
esprit@esprit:~$ ./script4
Addition: 13
Soustraction: 7
Multiplication: 30
Division: 3
Reste: 1
Parenthèses: 26
```

Expr : manipulation des chaînes

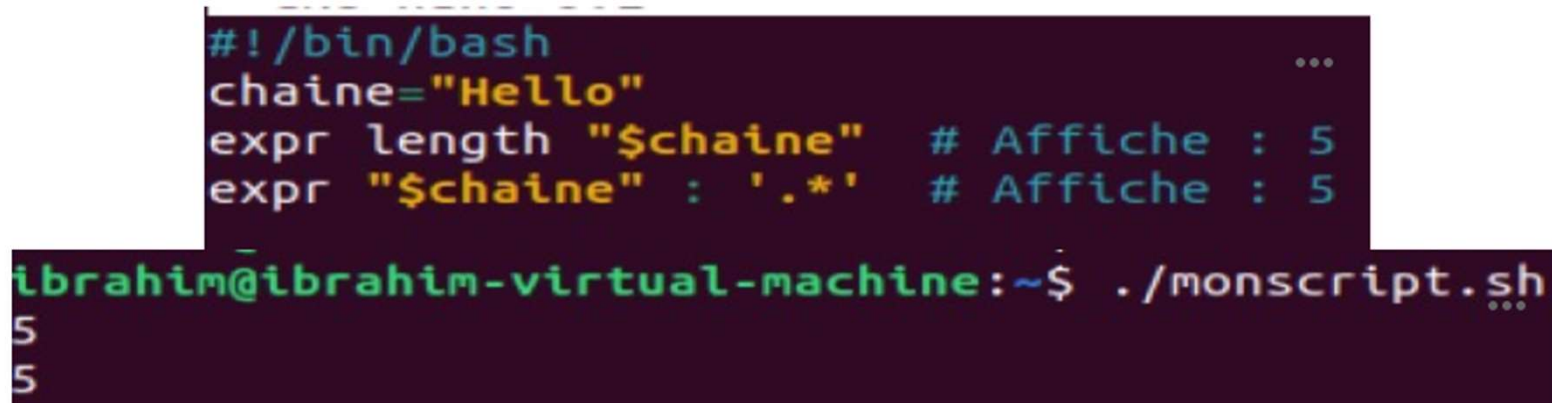
1- Longueur de chaînes de caractères

- `${#chaine}`
- `expr length $chaine`
- `expr "$chaine" : '.*'`



```
#!/bin/bash
chaine=BonjourEsprit2021

echo ${#chaine} # 15
echo `expr length $chaine` # 15
echo `expr "$chaine" : '.*'` # 15
```



```
#!/bin/bash
chaine="Hello"
expr length "$chaine" # Affiche : 5
expr "$chaine" : '.*' # Affiche : 5

ibrahim@ibrahim-virtual-machine:~$ ./monscript.sh
5
5
```

Expr : manipulation des chaînes

2- Longueur de sous-chaînes correspondant à un motif

➤ Au début d'une chaîne

`expr match "$chaîne" 'He' # Affiche : 2 (longueur de la correspondance)`

➤ Utiliser expr avec un motif :

`expr "$chaîne" : 'He.*' # Affiche : 5 (longueur de la correspondance)`

➤ Trouver la position d'une sous-chaîne dans une chaîne :

`expr index "$chaîne" 'l' # Affiche : 3 (première occurrence de 'l')`

```
#!/bin/bash
chaîne="bonjouresprit2024"
echo `expr match "$chaîne" 'bonjou'` #affiche 6 (longuer de la correspondance)
echo `expr "$chaîne" : 'bon[a-z]*.2'` # Affiche : 14(longueur de la correspondance)
echo `expr index "$chaîne" 'r'` # Affiche : 7 (première occurrence de 'l')
```

```
esprit@esprit:~$ ./script0
6
14
7
```

```
#!/bin/bash
chaîne="hello_2A"
expr match "$chaîne" 'he' #affiche : 2 (longuer de la correspondance)
expr "$chaîne" : 'he.*_' # Affiche :6 (longueur de la correspondance)
expr index "$chaîne" 'l' # Affiche : 3 (première occurrence de 'l')
```

```
esprit@esprit:~$ ./script4
2
6
3
```


Expr : manipulation des chaînes

3- Extraction d'une sous-chaîne

➤ Extraire une sous-chaîne à une position donnée :

`${chaîne:position:longueur}` #extrait \$longueur caractères d'une sous-chaîne de \$chaîne à la position \$position

```
#!/bin/bash
chaîne="bonjouesprit2025"
echo ${chaîne:0}
echo ${chaîne:5}
echo ${chaîne:7}
echo ${chaîne:3:7}
echo ${chaîne: -5}
echo ${chaîne:(-5)}
echo ${chaîne:2: -3}
```

```
esprit@esprit:~$ ./script4
bonjouesprit2025
uresprit2025
esprit2025
jouresp
t2025
t2025
njouesprit2
```

➤ Utiliser expr pour extraire une sous-chaîne à une position donnée :

`expr substr $chaîne position longueur` #extrait \$longueur caractères à partir de \$chaîne en commençant à \$position

```
#!/bin/bash
chaîne="bonjouesprit2025"
expr substr $chaîne 3 8
expr substr $chaîne 8 3
echo `expr substr $chaîne 8 3`
```

```
esprit@esprit:~$ ./script4
njouespr
esp
esp
```

Expr : manipulation des chaînes

- **Extrait \$souschaine à partir du début de \$chaine** : où \$souschaine est une expression rationnelle

expr "\$chaine" : '\(\$souschaine\)'

```
#!/bin/bash
chaine="bonjourEsprit2025"
expr match "$chaine" '\(bon\)'
expr match "$chaine" '\([a-z]*..\)'
expr "$chaine" : '\([a-z]*[A-Z]\)'
expr "$chaine" : '\(...*[0-9]\)'
expr "$chaine" : '\(.....\)'
```

```
esprit@esprit:~$ ./script4
bon
bonjourEs
bonjourE
bonjourEsprit2025
bonjourEs
```

- **Extrait \$souschaine à la fin de \$chaine** : où \$souschaine est une expression rationnelle.

expr "\$chaine" : '.\(\$souschaine\)'*

```
#!/bin/bash
chaine="bonjourEsprit2025"
expr "$chaine" : '.*\(...\)'
expr "$chaine" : '.*\([a-z]....\)'
```

```
esprit@esprit:~$ ./script4
2025
t2025
```

Expr : manipulation des chaînes

4- Suppression de sous-chaînes

- *Supprimer la correspondance la plus petite de \$souschaine à partir du début de \$chaine :*

`${chaine#souschaine}`

- *Supprimer la correspondance la plus grande de \$souschaine à partir du début de \$chaine :*

`${chaine##souschaine}`

```
#!/bin/bash
chaine="bonjourEsprit2025"
echo ${chaine#b*r} #supprime la plus petite correspondance entre b et r.
echo ${chaine##b*r} #supprime la plus grande correspondance entre b et r.
```

```
esprit@esprit:~$ ./script4
Esprit2025
it2025
```

Expr : manipulation des chaînes

4- Suppression de sous-chaînes

- Supprimer la plus petite correspondance de \$souschaine à partir de la fin de \$chaine

`${chaine%souschaine}`

- Supprimer la correspondance la plus grande de \$souschaine à partir de la fin de \$chaine

`${chaine%%souschaine}`

```
#!/bin/bash
chaine="bonjourEsprit2025"
echo ${chaine%r*5} #supprime la plus petite correspondance entre r et 5 à partir de la fin de chaine.
echo ${chaine%%r*5} #supprime la plus grande correspondance entre r et 5 à partir de la fin de chaine
```

```
esprit@esprit:~$ ./script4
bonjourEsp
bonjou
```

Expr : manipulation des chaînes

5- Remplacement de sous-chaîne

- *Remplacer la première correspondance de souschaîne par remplacement*

`${chaîne/souschaîne/remplacement}`

- *Remplacer toutes les correspondances de souschaîne avec remplacement*

`${chaîne//souschaîne/remplacement}`

```
#!/bin/bash
chaîne="bonjourEspritEsprit2025"
echo ${chaîne/Esp/abc} #remplacer la première correspondance de Esp par abc.
echo ${chaîne//Esp/abc} #remplacer toutes les correspondance de Esp par abc.
```

```
esprit@esprit:~$ ./script4
bonjourabcritEsprit2025
bonjourabcritabcrit2025
```