
Module : Système et Scripting

Année Universitaire
2023-2024

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

Systeme & Scripting

Chapitre II : Le langage

Plan

1. Shell : Le principe des scripts
2. Fonctionnement des scripts
3. Exécution des scripts
4. composants des scripts
5. Déclaration des variables
6. echo : affichage des variables
7. read : saisie des variables
8. Les variables d'environnement
9. Les variables spéciales
10. Tableaux
11. Expr : arithmétique
12. Expr : manipulation des chaînes

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 :

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

- Si le script est exécutable il suffit passer le script en argument à un shell "bash monscript.sh".
- On peut aussi utiliser l'opérateur "." " ./monscript.sh".

composants des scripts

- ❖ une commande shell de base : find, cut ou echo
- ❖ une variable : \$var
- ❖ un commentaire : #ceci est un commentaire
- ❖ une fonction : définition d'un sous programme ou procédure
- ❖ une structure de contrôle : if-then-else, for ou while

SYNTAXE

- ❖ # 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.
- ❖ ; 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 :

```
$ vim 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
```

echo : affichage des variables

- La commande echo permet d'afficher une ligne.

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

- lors de insertion des retours à la ligne, il faudra activer le paramètre -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
```

echo : affichage des variables

❖ 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)

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
```

- ❖ les doubles quotes demandent à bash d'analyser le contenu du message
S'il trouve des symboles spéciaux (comme des variables), il les interprète.

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

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

- ❖ les back quotes 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
```

read : saisie des variables

- 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 "
```

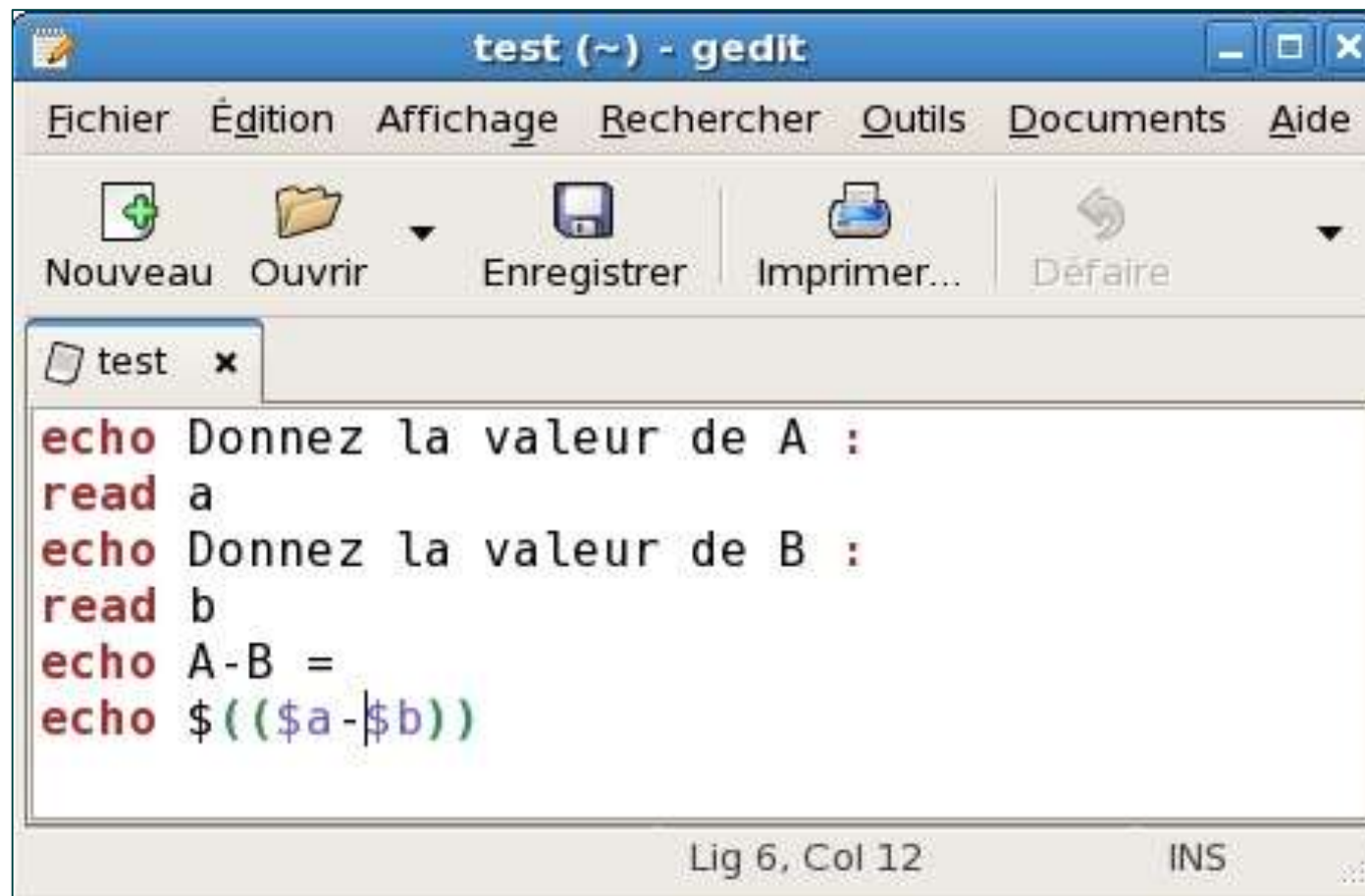
- Affecter simultanément une valeur à plusieurs variables

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

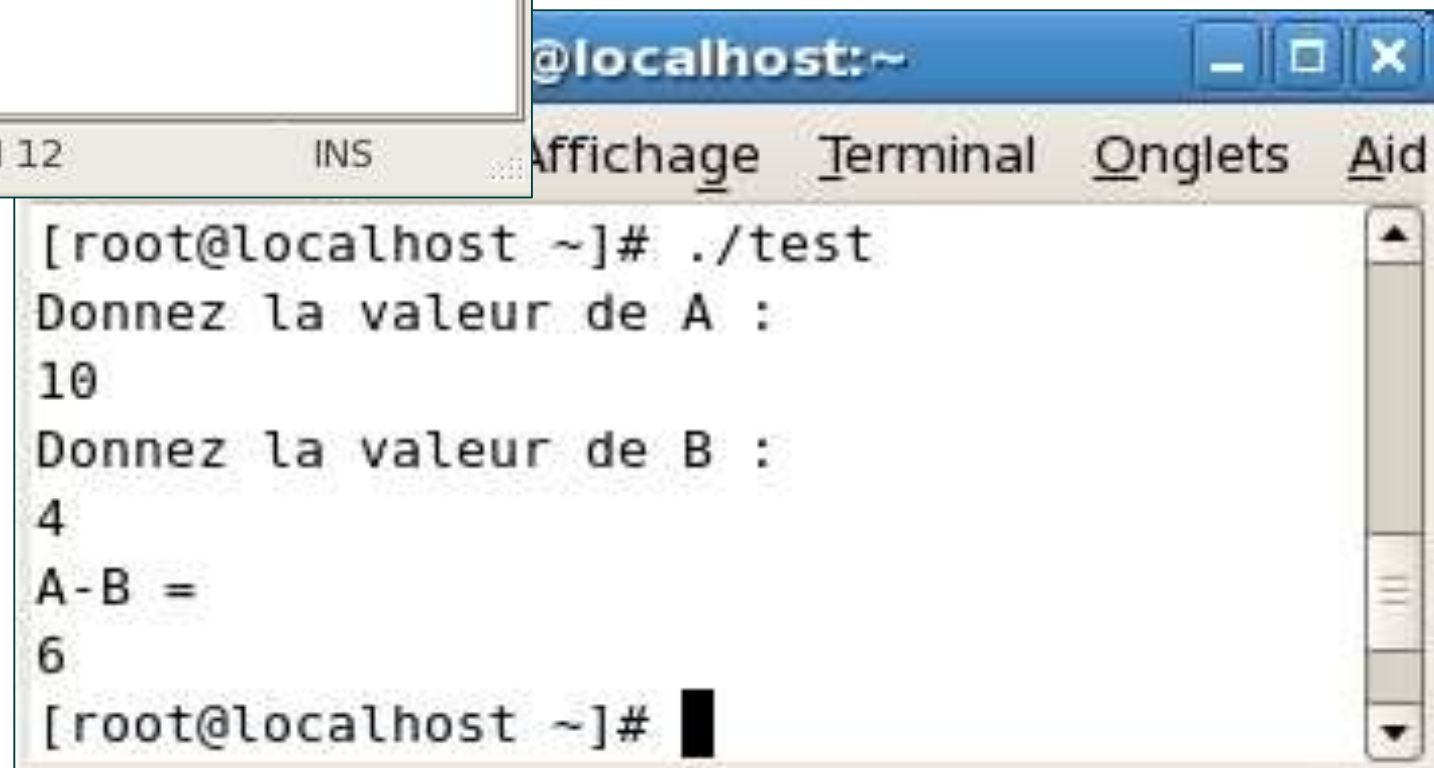
- afficher un message de prompt

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

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
```



```
@localhost:~
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

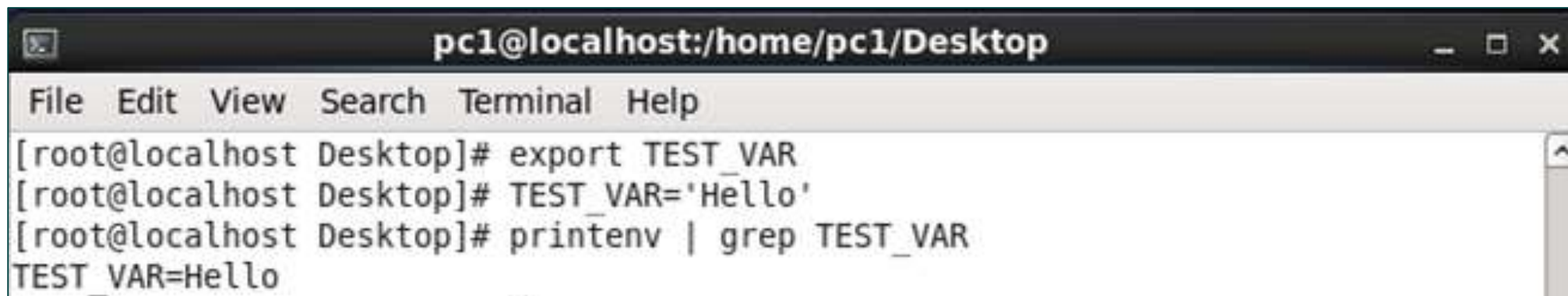
- SHELL : indique quel type de shell est en cours d'utilisation (sh, bash, ksh...);
- PATH : une liste des répertoires qui contiennent des exécutables que vous souhaitez pouvoir lancer sans indiquer leur répertoire.
- EDITOR : l'éditeur de texte par défaut qui s'ouvre lorsque cela est nécessaire ;
- 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
```

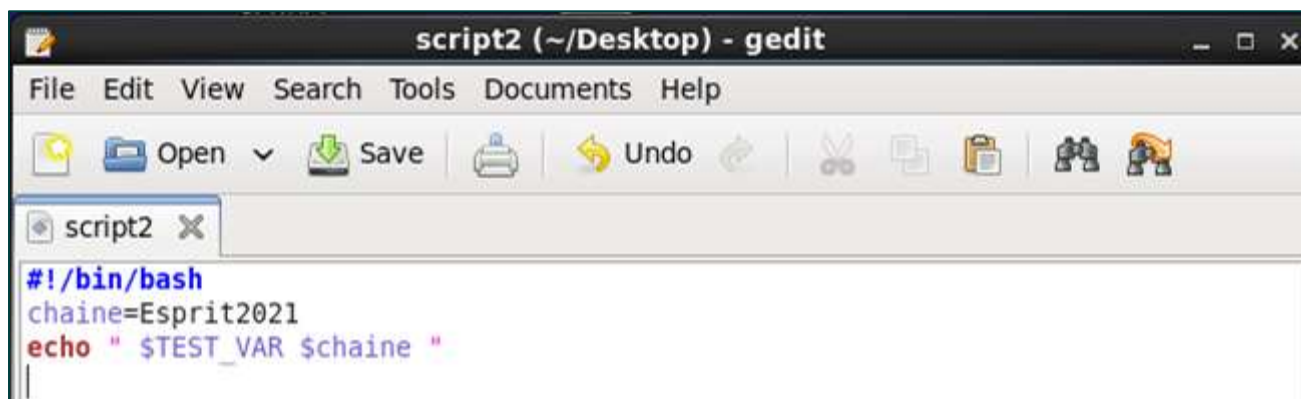
Les variables d'environnement

- Création des variables d'environnement

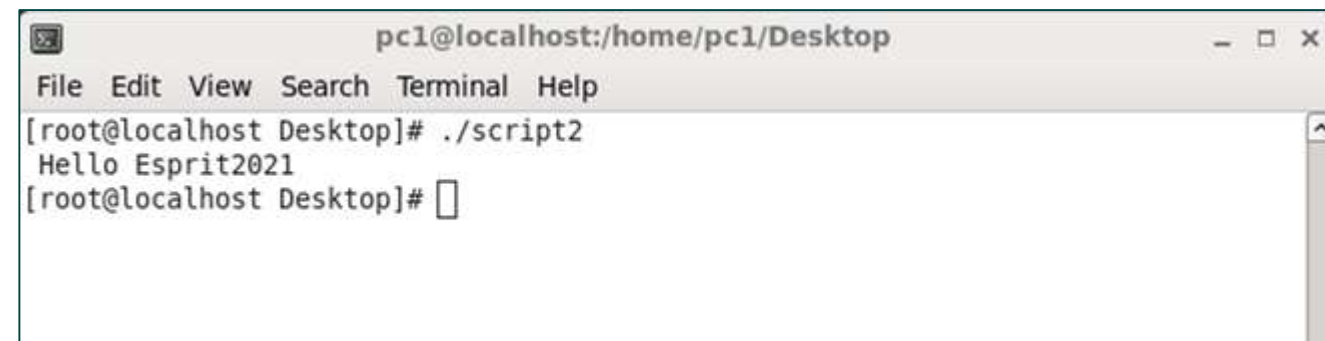


```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# export TEST_VAR
[root@localhost Desktop]# TEST_VAR='Hello'
[root@localhost Desktop]# printenv | grep TEST_VAR
TEST_VAR=Hello
```

- 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]#
```

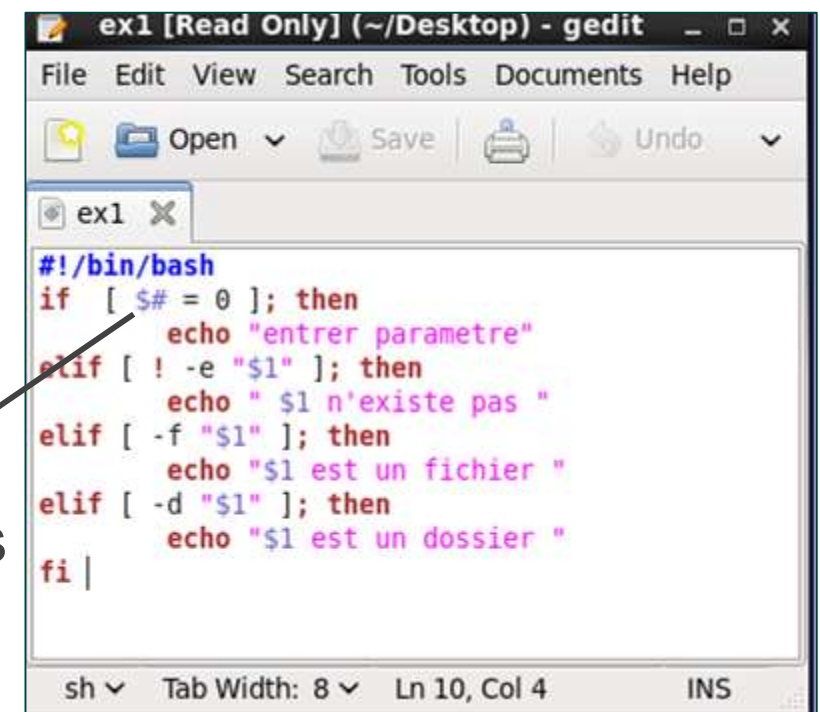
Les variables spéciales

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

<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



```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[pc1@localhost Desktop]$ su
Password:
[root@localhost Desktop]# ./ex1 untitled/
```



```
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
```

`$0` Le 1^{er} argument `$1` Le Nombre d'arguments

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

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

Tableaux

- 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
```

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

```
$ echo ${tab[0]}  
10  
$ echo ${tab[2]}  
12
```

Tableaux

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

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

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

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

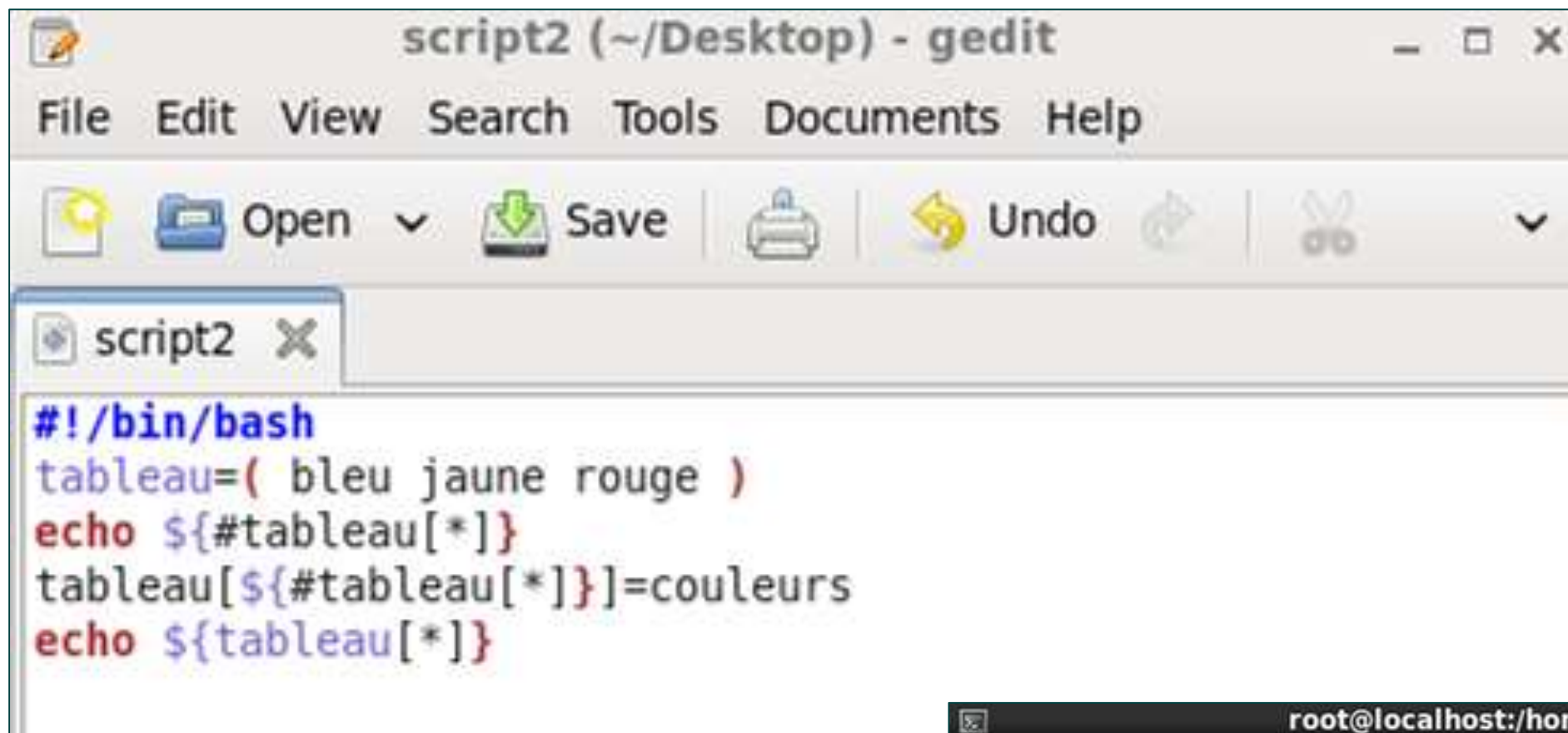
```
${#nomtableau[*]}
```

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

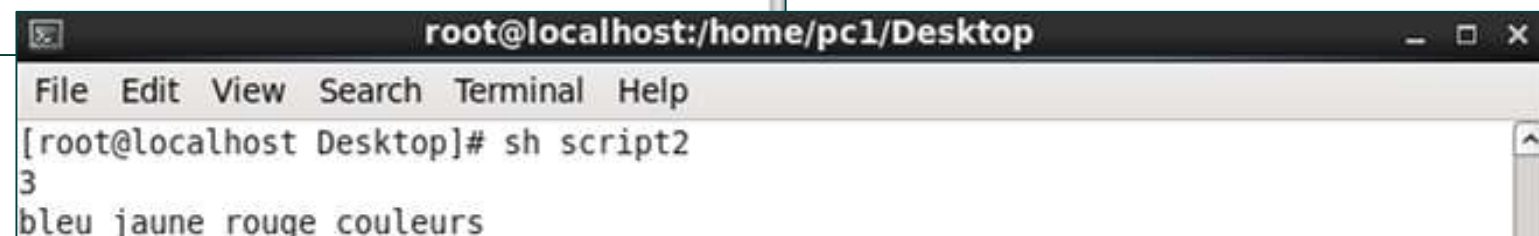
Tableaux

- Ajout d'un élément à un tableau:

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



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



```
root@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
3
bleu jaune rouge couleurs
```

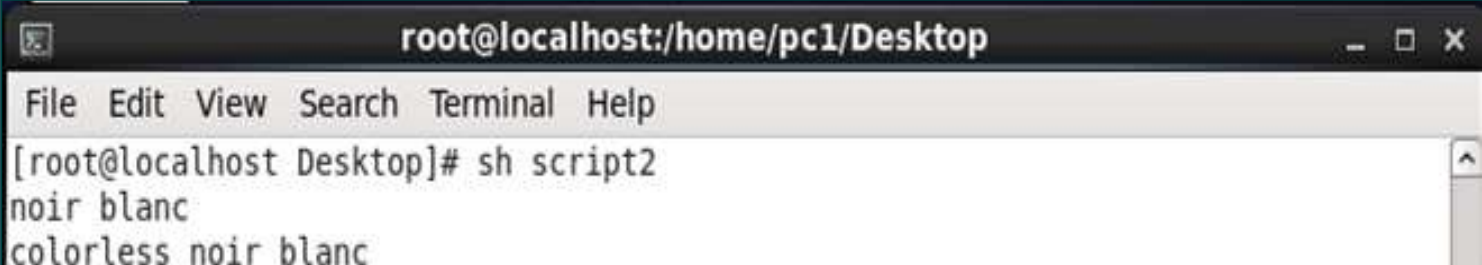

Tableaux

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

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



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



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

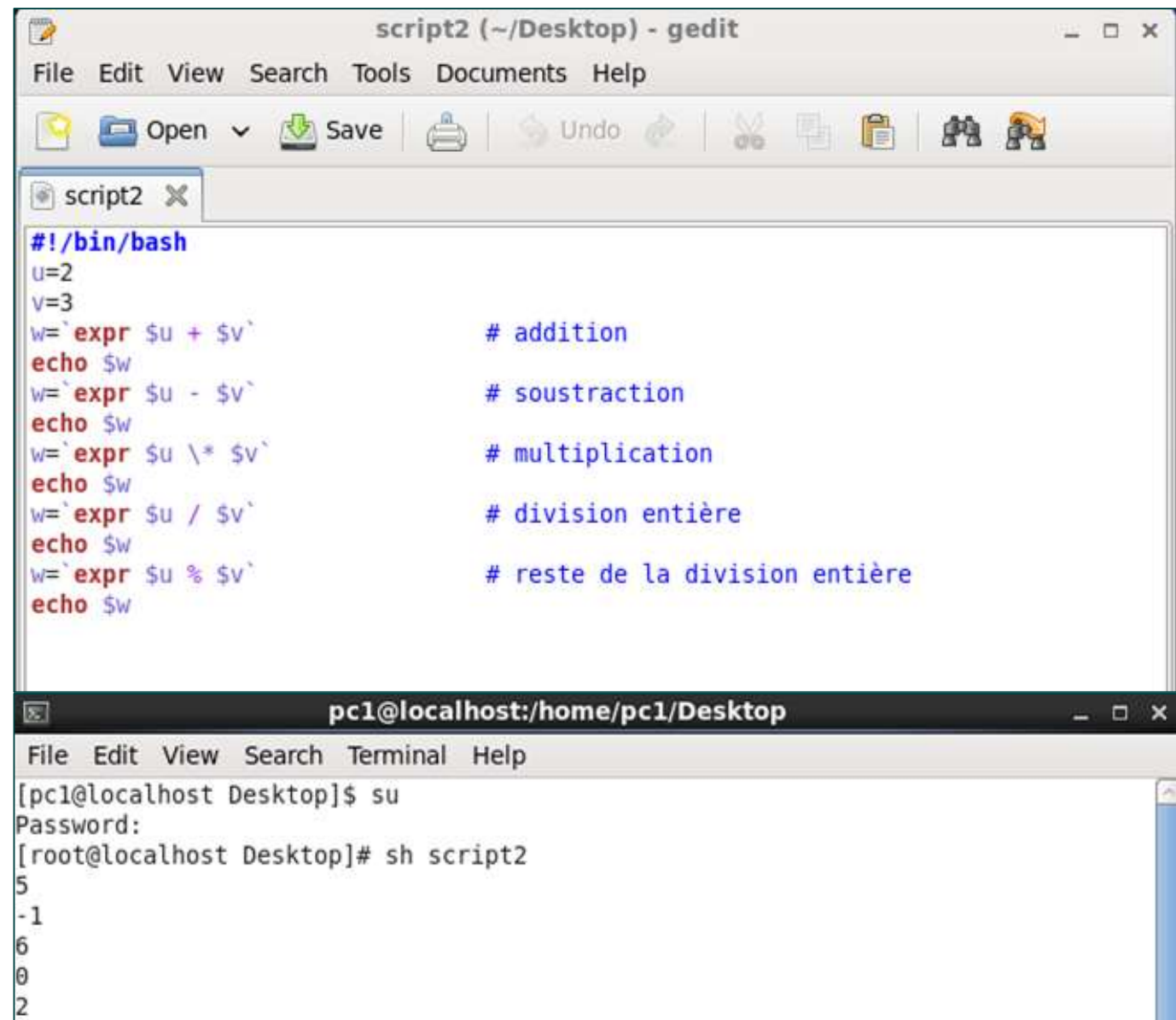
- ❖ supprimer un tableau :
- ❖ supprimer la case d'un tableau :

```
unset nom-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.



The image shows two overlapping windows. The top window is a Gedit text editor titled 'script2 (~/.Desktop) - gedit'. It contains a bash script with the following content:

```
#!/bin/bash
u=2
v=3
w=`expr $u + $v`           # addition
echo $w
w=`expr $u - $v`           # soustraction
echo $w
w=`expr $u \* $v`          # multiplication
echo $w
w=`expr $u / $v`           # division entière
echo $w
w=`expr $u % $v`           # reste de la division entière
echo $w
```

The bottom window is a terminal titled 'pc1@localhost:/home/pc1/Desktop'. It shows the execution of the script:

```
[pc1@localhost Desktop]$ su
Password:
[root@localhost Desktop]# sh script2
5
-1
6
0
2
```

Expr : manipulation des chaînes

- Longueur de chaînes de caractères

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



```
script2 (~/Desktop) - gedit
File Edit View Search Tools Documents Help
script2
#!/bin/bash
chaine=BonjourEsprit2021

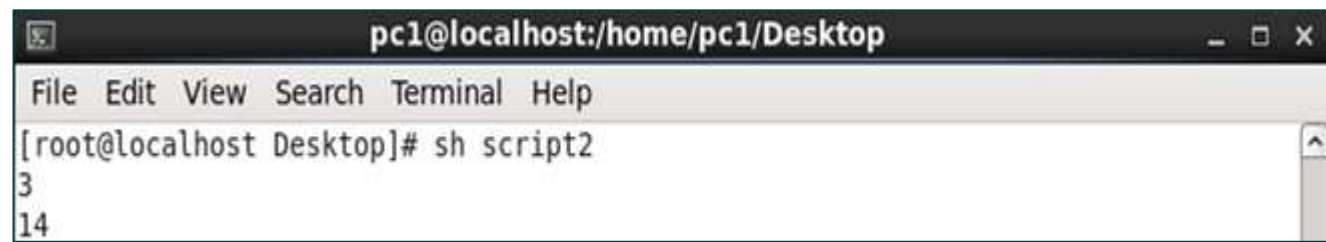
echo ${#chaine} # 15
echo `expr length $chaine` # 15
echo `expr "$chaine" : '.*'` # 15
```

- ❖ Longueur de sous-chaînes correspondant à un motif au début d'une chaîne

- `expr match "$chaine" '$souschaine'`
- `expr "$chaine" : '$souschaine'`



```
script2 (~/Desktop) - gedit
File Edit View Search Tools Documents Help
script2
#!/bin/bash
chaine=BonjourEsprit2021
echo `expr match "$chaine" 'Bon'`
echo `expr "$chaine" : 'Bon[A-Z]*.2'`
```




```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
3
14
```

Expr : manipulation des chaînes

- `expr index $chaine $souschaine`
 - Position numérique dans `$chaine` du premier caractère dans `$souschaine` qui correspond.

echo `expr index "\$chaine" 2`
The script is saved in a file named 'script2' on the desktop." data-bbox="8 507 484 789"/>

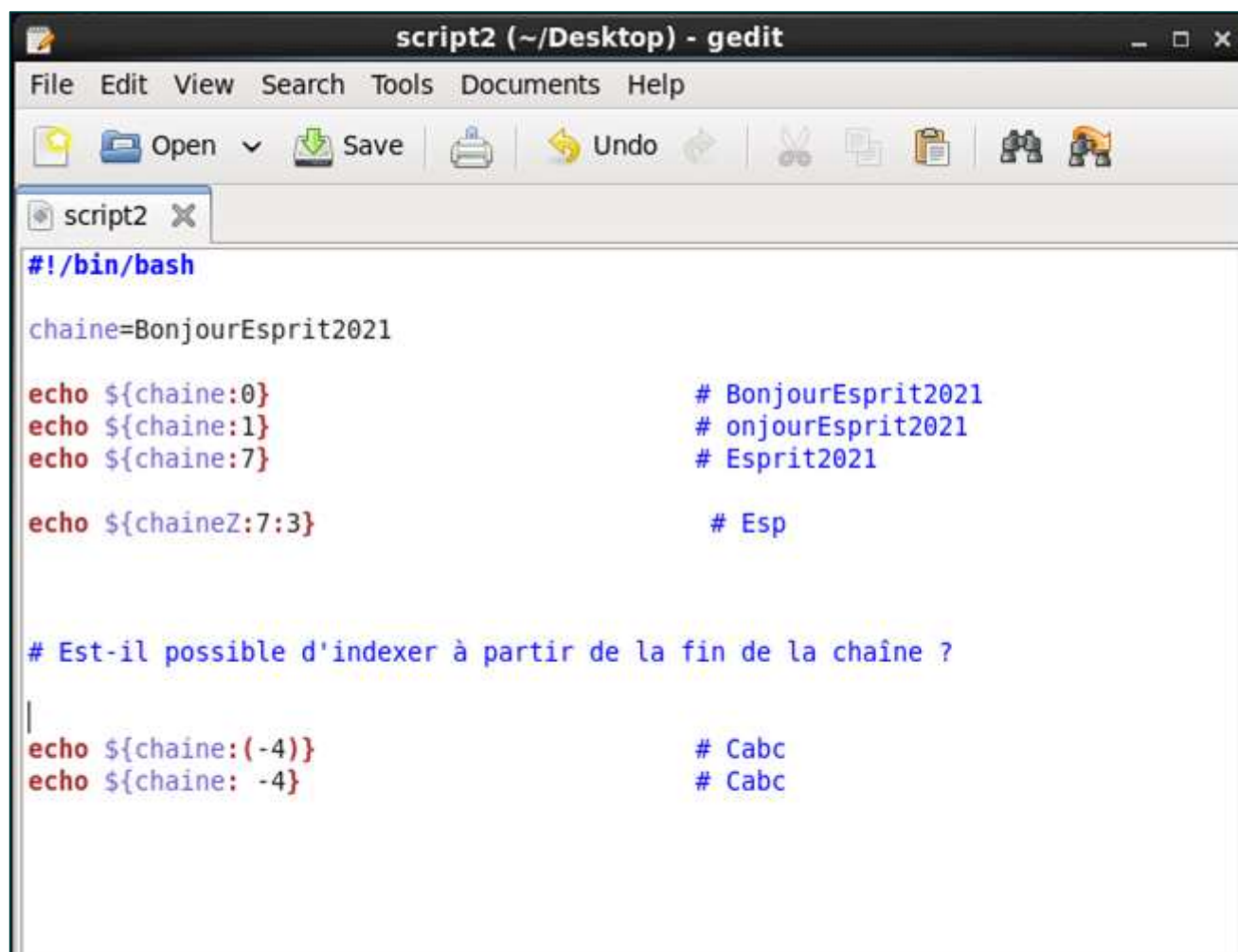
```
#!/bin/bash  
chaine=BonjourEsprit2021  
echo `expr index "$chaine" Es`  
echo `expr index "$chaine" 2`
```



```
pc1@localhost:/home/pc1/Desktop  
[root@localhost Desktop]# sh script2  
8  
14
```

Expr : manipulation des chaînes

- ❖ Extraction d'une sous-chaîne
 - `${chaine:position:longueur}`
 - Extrait \$longueur caractères d'une sous-chaîne de \$chaine à la position \$position.



```
#!/bin/bash

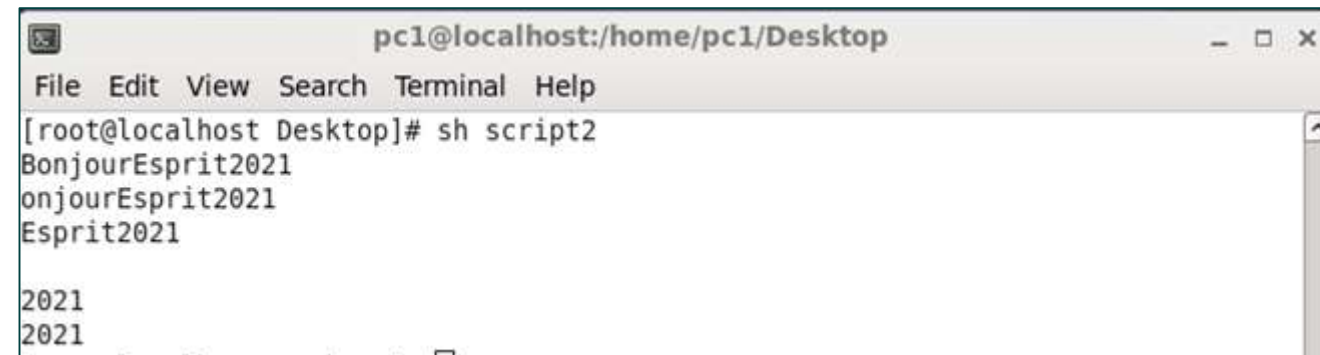
chaine=BonjourEsprit2021

echo ${chaine:0}           # BonjourEsprit2021
echo ${chaine:1}           # onjourEsprit2021
echo ${chaine:7}           # Esprit2021

echo ${chaineZ:7:3}        # Esp

# Est-il possible d'indexer à partir de la fin de la chaîne ?

echo ${chaine:(-4)}        # Cabc
echo ${chaine: -4}         # Cabc
```



```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
BonjourEsprit2021
onjourEsprit2021
Esprit2021

Esp

Cabc
Cabc
```


Expr : manipulation des chaînes

- ❖ Extraction d'une sous-chaîne
 - `expr substr $chaine $position $longueur`
 - Extrait \$longueur caractères à partir de \$chaine en commençant à \$position.

A screenshot of a gedit editor window titled "script2 (~/Desktop) - gedit". The window shows a shell script with the following content:

```
#!/bin/bash  
chaine=BonjourEsprit2021  
echo `expr substr $chaine 1 7`  
echo `expr substr $chaine 8 6`
```

A screenshot of a terminal window titled "pc1@localhost:/home/pc1/Desktop". The terminal shows the execution of the script "script2", resulting in the output "Bonjour" and "Esprit".

```
pc1@localhost:/home/pc1/Desktop  
[root@localhost Desktop]# sh script2  
Bonjour  
Esprit
```

Expr : manipulation des chaînes

- `expr "$chaine" : '\($souschaine\)'`
 - Extrait `$souschaine` à partir du début de `$chaine`, et où `$souschaine` est une expression rationnelle.

A screenshot of a gedit editor window titled "script2 (~/Desktop) - gedit". The window shows a shell script with the following content:

```
#!/bin/bash
chaine=BonjourEsprit2021
echo `expr match "$chaine" '\(Bon\)\'`
echo `expr match "$chaine" '\([a-z]*[A-Z]...\)`
echo `expr "$chaine" : '\([a-z]*[A-Z]...[0-9]\)`
echo `expr "$chaine" : '\(.....\)`
```

A screenshot of a terminal window titled "pc1@localhost:/home/pc1/Desktop". The terminal shows the execution of the script "script2", which outputs:

```
[root@localhost Desktop]# sh script2
Bon
BonjourEsprit202
BonjourEsprit2021
Bonjour
```

Expr : manipulation des chaînes

- `expr "$chaine" : '.*\($souschaine\)'`
 - Extrait `$souschaine` à la fin de `$chaine`, et où `$souschaine` est une expression rationnelle.

The script is saved in a file named 'script2' on the desktop." data-bbox="20 539 552 818"/>

```
#!/bin/bash  
chaine=BonjourEsprit2021  
echo `expr "$chaine" : '.*\($souschaine\)'`
```



```
pc1@localhost:/home/pc1/Desktop  
[root@localhost Desktop]# sh script2  
it2021
```


Expr : manipulation des chaînes

❖ Suppression de sous-chaînes

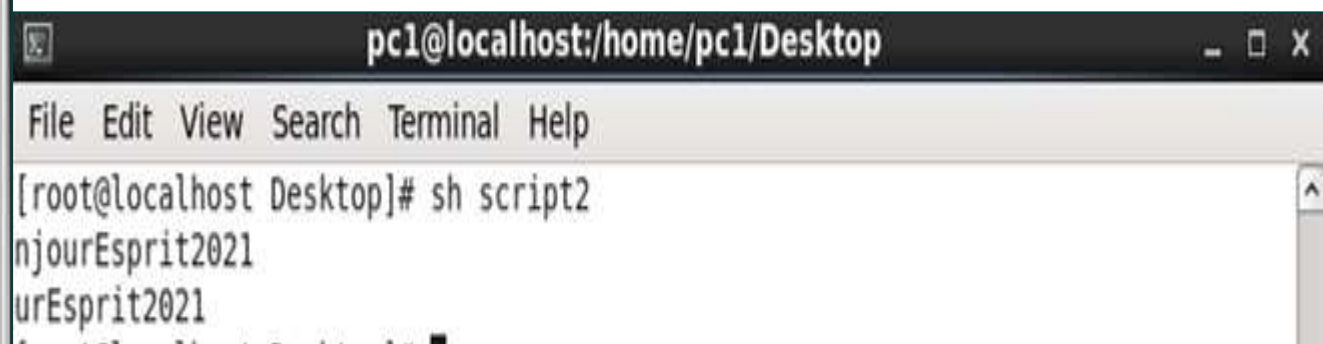
- `${chaine#souschaine}`
 - Supprime la correspondance la plus petite de \$souschaine à partir du début de \$chaine.
- `${chaine##souschaine}`
 - Supprime la correspondance la plus grande de \$souschaine à partir du début de \$chaine.



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

echo ${chaine#B*o}
# Supprime la plus petite correspondance entre 'B' et 'o'.

echo ${chaine##B*o}
# Supprime la plus grande correspondance entre 'B' et 'o'|
```



```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
njourEsprit2021
urEsprit2021
```

Expr : manipulation des chaînes

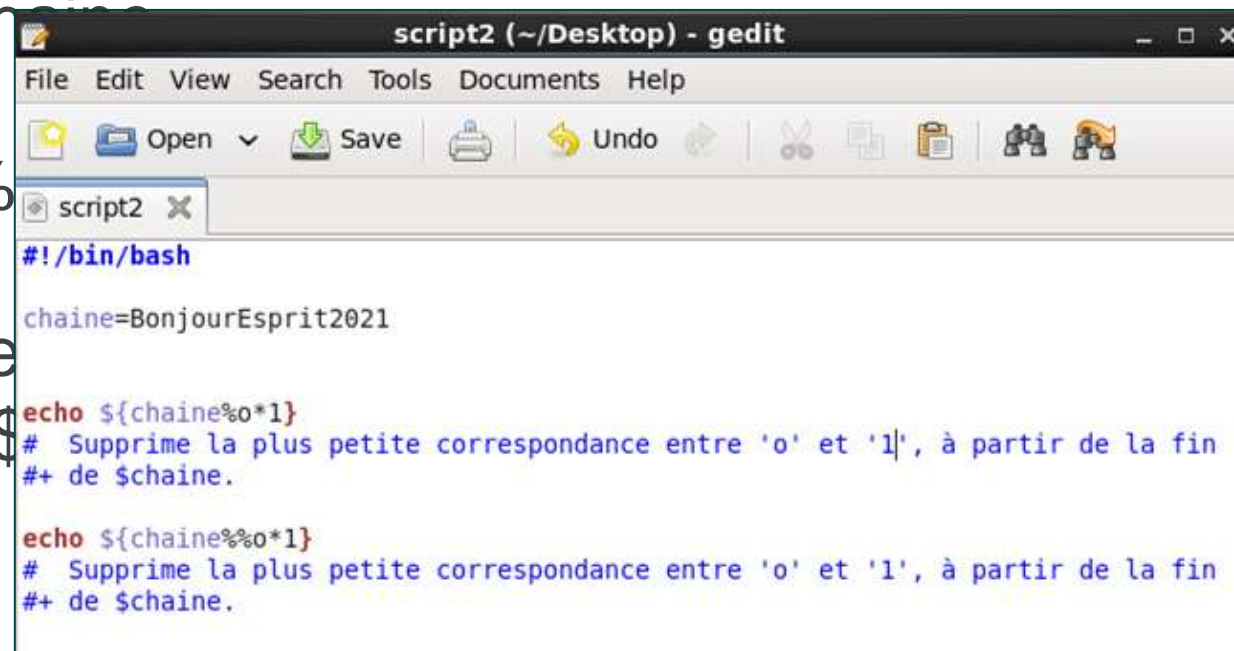
❖ Suppression de sous-chaînes

➤ `${chaine%souschaine}`

➤ Supprime la plus petite correspondance de \$souschaine à partir de la fin de \$chaine.

➤ `${chaine%%o*1}`

➤ Supprime la plus petite correspondance entre 'o' et '1', à partir de la fin de \$chaine.



```
#!/bin/bash
chaine=BonjourEsprit2021
echo ${chaine%o*1}
# Supprime la plus petite correspondance entre 'o' et '1', à partir de la fin
# de $chaine.
echo ${chaine%%o*1}
# Supprime la plus petite correspondance entre 'o' et '1', à partir de la fin
# de $chaine.
```



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

Expr : manipulation des chaînes

❖ Remplacement de sous-chaîne

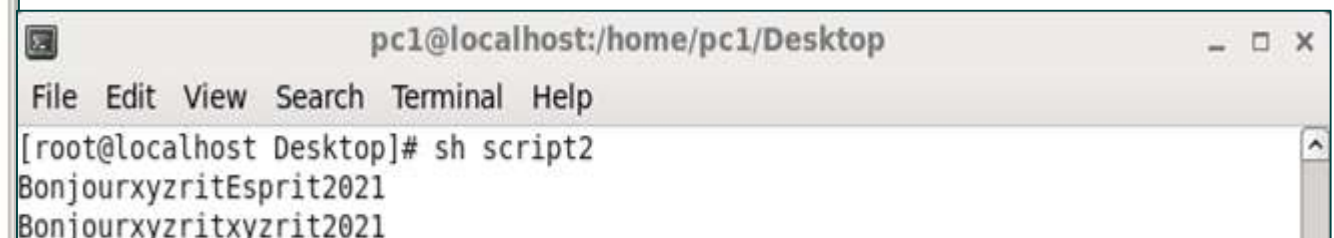
- `${chaine/souschaine/remplacement}`
 - Remplace la première correspondance de `$souschaine` par `$remplacement`.
- `${chaine//souschaine/remplacement}`
 - Remplace toutes les correspondances de `$souschaine` avec `$remplacement`.



```
#!/bin/bash
chaine=BonjourEspritEsprit2021

echo ${chaine/Esp/xyz}      # xyzABC123ABCabc
                           # Remplace la première correspondance de
                           #+ 'Esp' avec 'xyz'.

echo ${chaine//Esp/xyz}     # xyzABC123ABCxyz|
                           # Remplace toutes les correspondances de
                           #+ 'Esp' avec 'xyz'.
```



```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
BonjourxyzritEsprit2021
Bonjourxyzritxyzrit2021
```