
Administration & Sécurité des Systèmes d'Exploitation UNIX

Année Universitaire
2022-2023

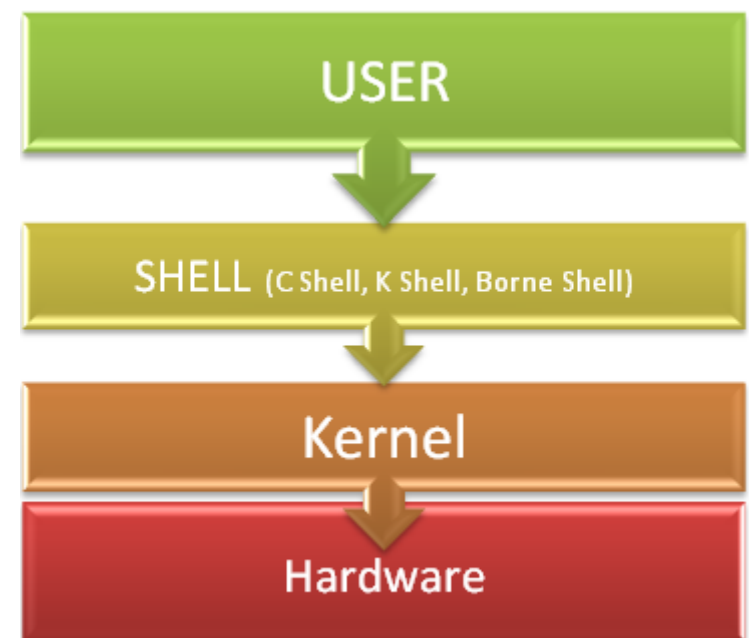
ASSEU

Chapitre V : Programmation Shell

Shell

Qu'est-ce qu'un shell ?

- Un shell est un programme qui fournit une interface à l'utilisateur pour utiliser les services du système d'exploitation.
- Shell accepte les commandes par l'utilisateur et les convertit en quelque chose que le noyau peut comprendre.



Shell

Il existe différents types d'interpréteurs de commandes « Shell » :

- ✓ sh: Bourne Shell
- ✓ bash: Bourne again Shell
- ✓ ksh: Korn Shell
- ✓ csh: C Shell
- Pour lister tous les shells disponibles dans votre système:
\$ cat /etc/shells
- Pour installer un nouveau shell:
yum install ksh
apt-get install ksh
- Pour changer le shell on exécute commande: **\$ chsh**

Script Shell

Les scripts Shell sont utilisés:

- ❖ 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.

Remarque: un script est un programme interprété et non compilé.

Script Shell

- ❖ Un script shell est un fichier qui contient:
 - ✓ Commandes shell
 - ✓ Variables
 - ✓ Structure de contrôle
 - ✓ Fonctions
- ❖ Ajout d'une ligne "shebang" :

`#!/bin/bash`

`#!/bin/sh`

la ligne « shebang » indique au système d'exploitation que ce fichier est un script et spécifie l'interpréteur utilisé pour exécuter ce script.

Script Shell

- Ajouter le droit d'exécution :
\$ chmod +x your-script-name
- Exécuter un script :
\$ bash your-script-name
\$ sh your-script-name
\$./your-script-name
- Pour déboguer un script :
\$ bash -x test.sh
- Pour exécuter un script depuis n'importe quel répertoire sans (./) devant, il faut le placer dans l'un des répertoires du PATH
echo \$ PATH

Déclaration des variables

- ❖ Création d'un nouveau script **variables.sh** :

```
$ vim variables.sh
```

- ❖ Indiquer le shell utilisé pour interpréter le script .

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

- ❖ Définition des variables.

- Créez une nouvelle variable en utilisant la syntaxe suivante
- **Syntaxe : `variable_nom= variable_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 l'**insertion des retours à la ligne**, il faut 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 message est : \$message

Pourquoi la variable message n'est-il pas analysé ?

- 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 :

```
#!/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 "
```

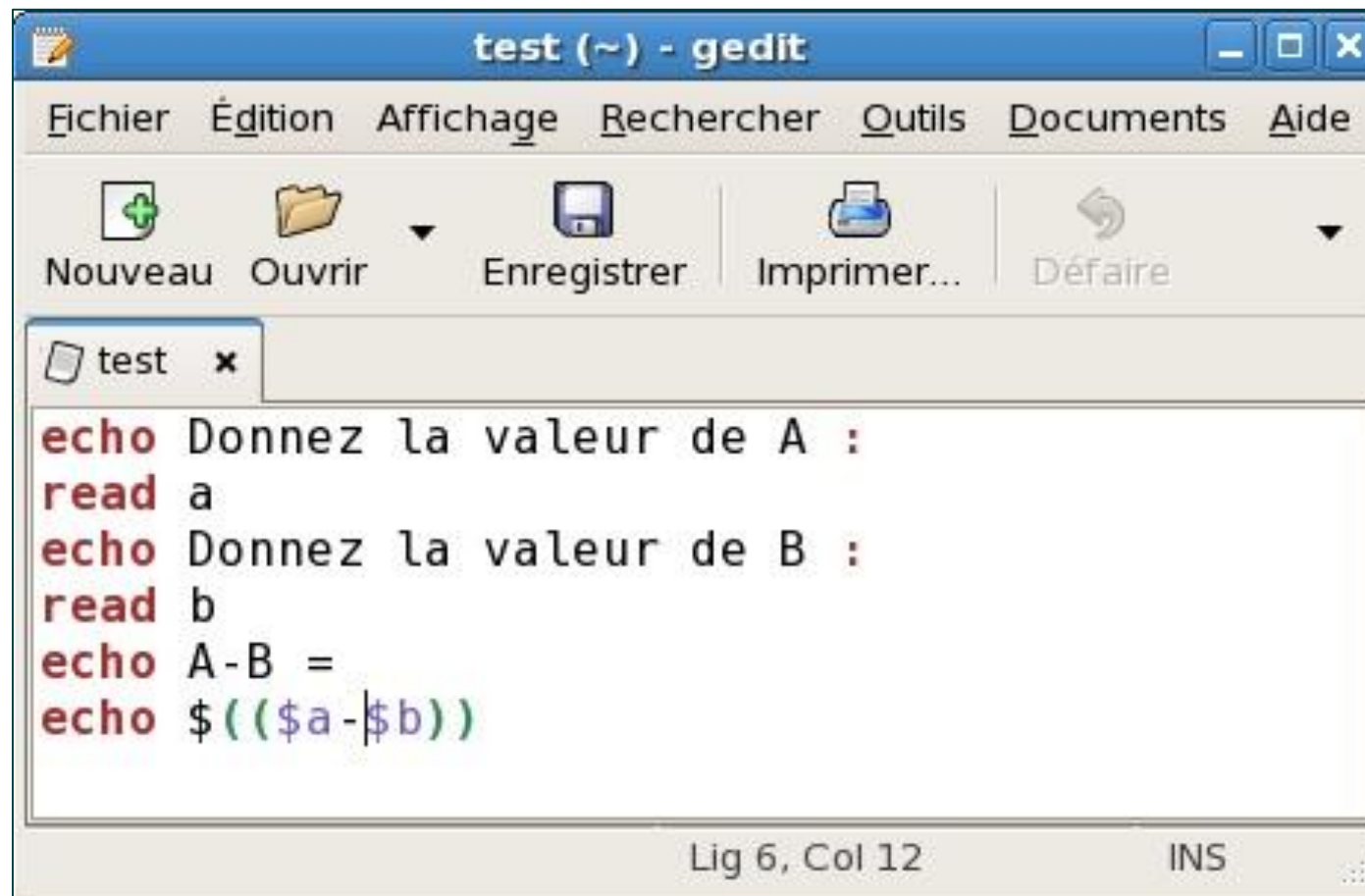
- ❖ Affecter simultanément une valeur à plusieurs variables

```
#!/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
```



```
root@localhost:~
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:/u
sr/sbin: PWD=/home/mateo21/bin
EDITOR=nano
HOME=/home/esprit
```

Les variables d'environnement

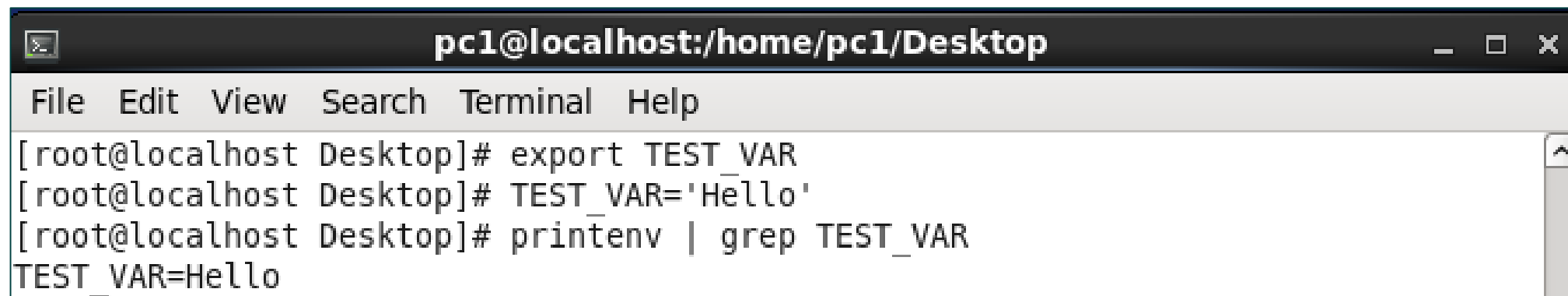
- SHELL : indique quel type de shell est en cours d'utilisation (sh, bash, ksh...);
- PATH : une liste des répertoires qui contiennent des exécutables que vous souhaitez pouvoir lancer sans indiquer leur répertoire.
- EDITOR : l'éditeur de texte par défaut qui s'ouvre lorsque cela est nécessaire ;
-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

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

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

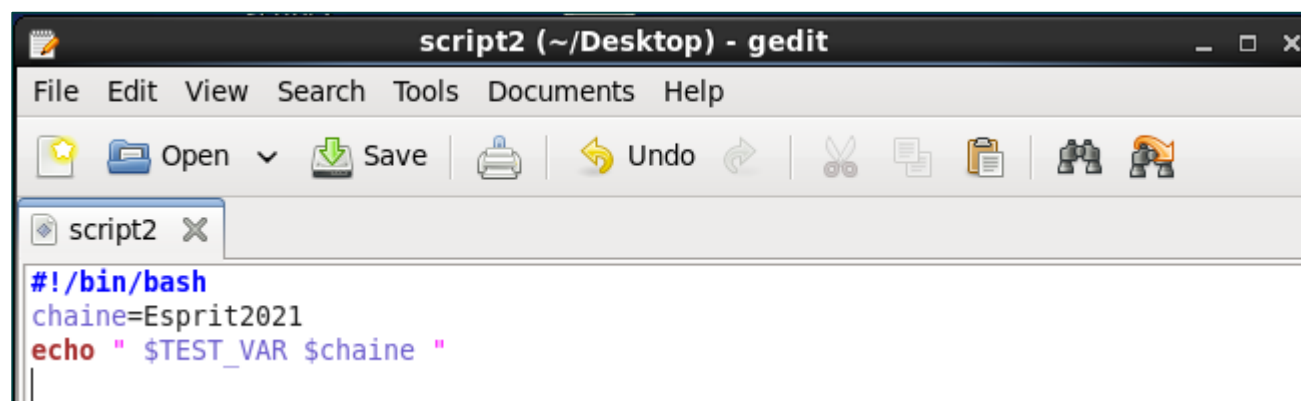
Les variables d'environnement

❖ Création des variables d'environnement

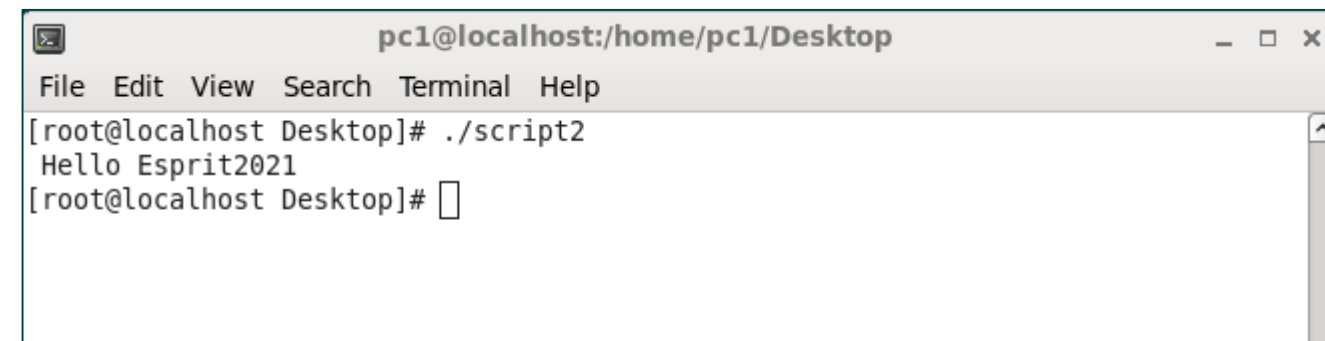


```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# export TEST_VAR
[root@localhost Desktop]# TEST_VAR='Hello'
[root@localhost Desktop]# printenv | grep TEST_VAR
TEST_VAR=Hello
```

❖ 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 "
```

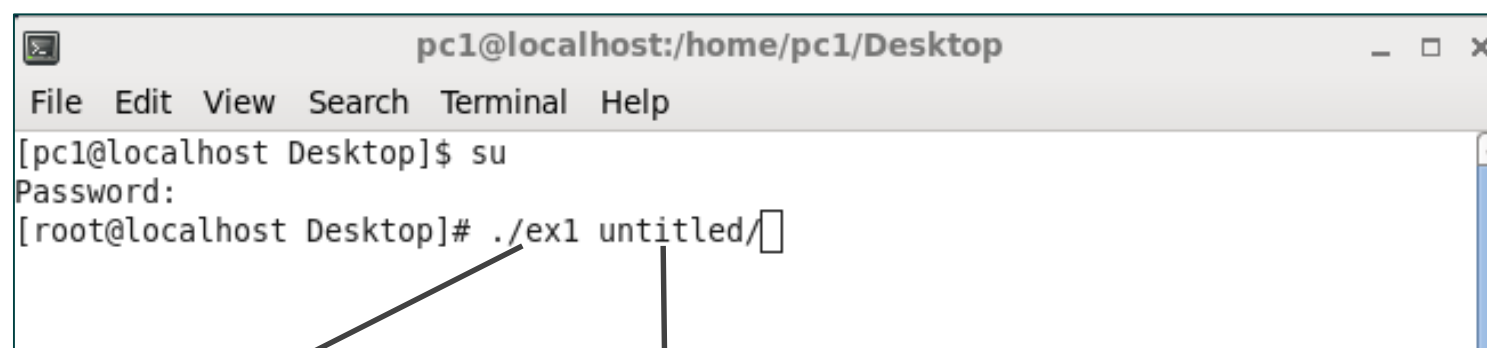


```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# ./script2
Hello Esprit2021
[root@localhost Desktop]#
```


Les variables spéciales

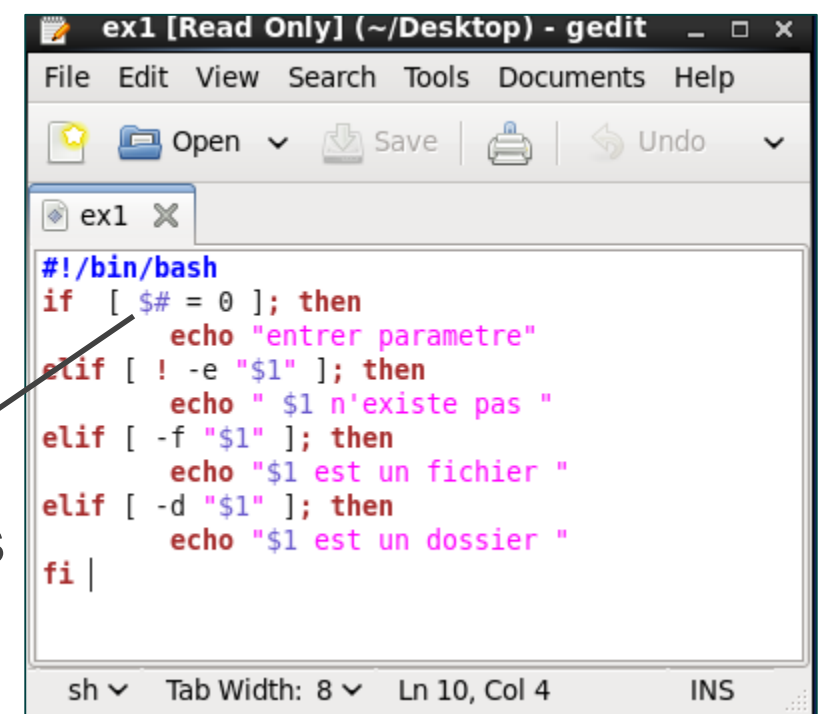
- ❖ Le script shell récupère les arguments de la ligne de commande dans des variables réservées appelées paramètres positionnels:

<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/
```

The terminal window shows the execution of the script `./ex1` with the argument `untitled/`. Arrows from the text below point to `$0` (the script name) and `$1` (the first argument).



```
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
```

The script editor window shows the code of the script `ex1`. The code checks the number of arguments (`$#`) and the type of the first argument (`$1`).

`$0`

Le 1^{er} argument `$1`

Le Nombre d'arguments

Tableaux

❖ Plusieurs méthodes pour créer un tableau

➤ 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 des 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 :

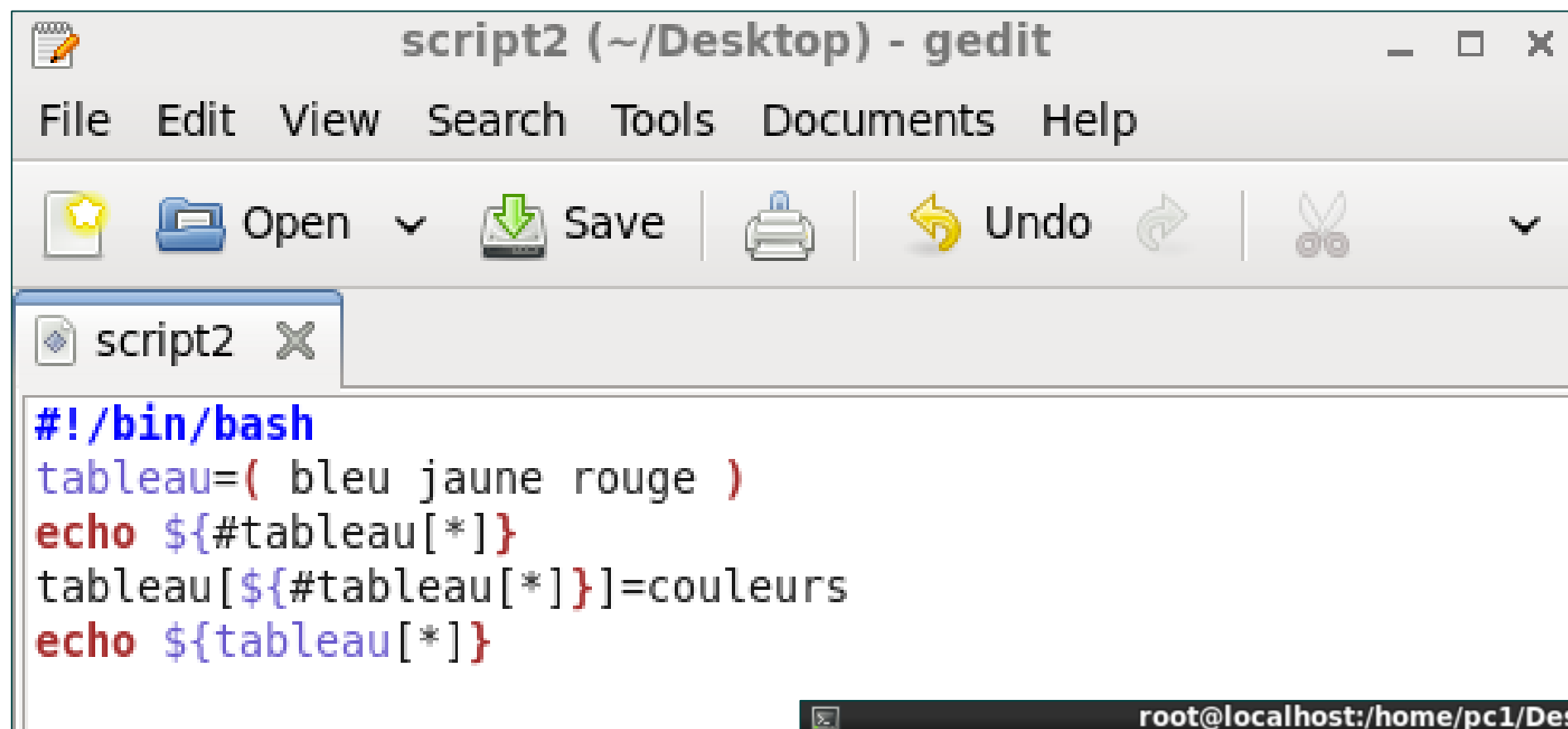
```
${#nomtableau[*]}
```

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

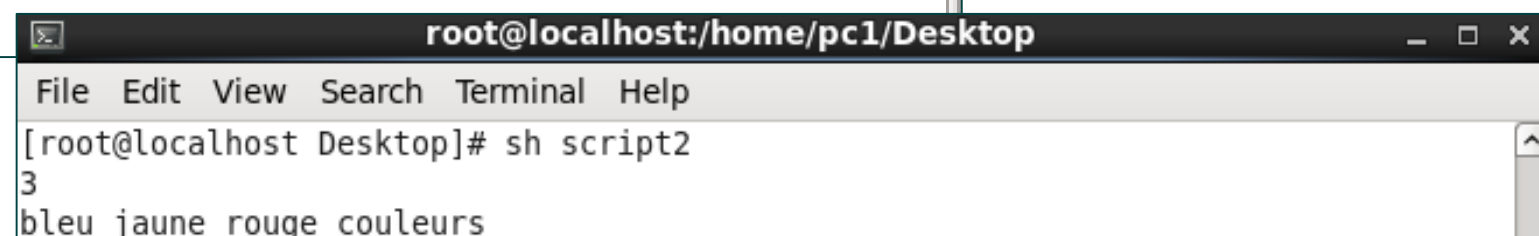
Tableaux

- ❖ Ajout d'un élément à un tableau:

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



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

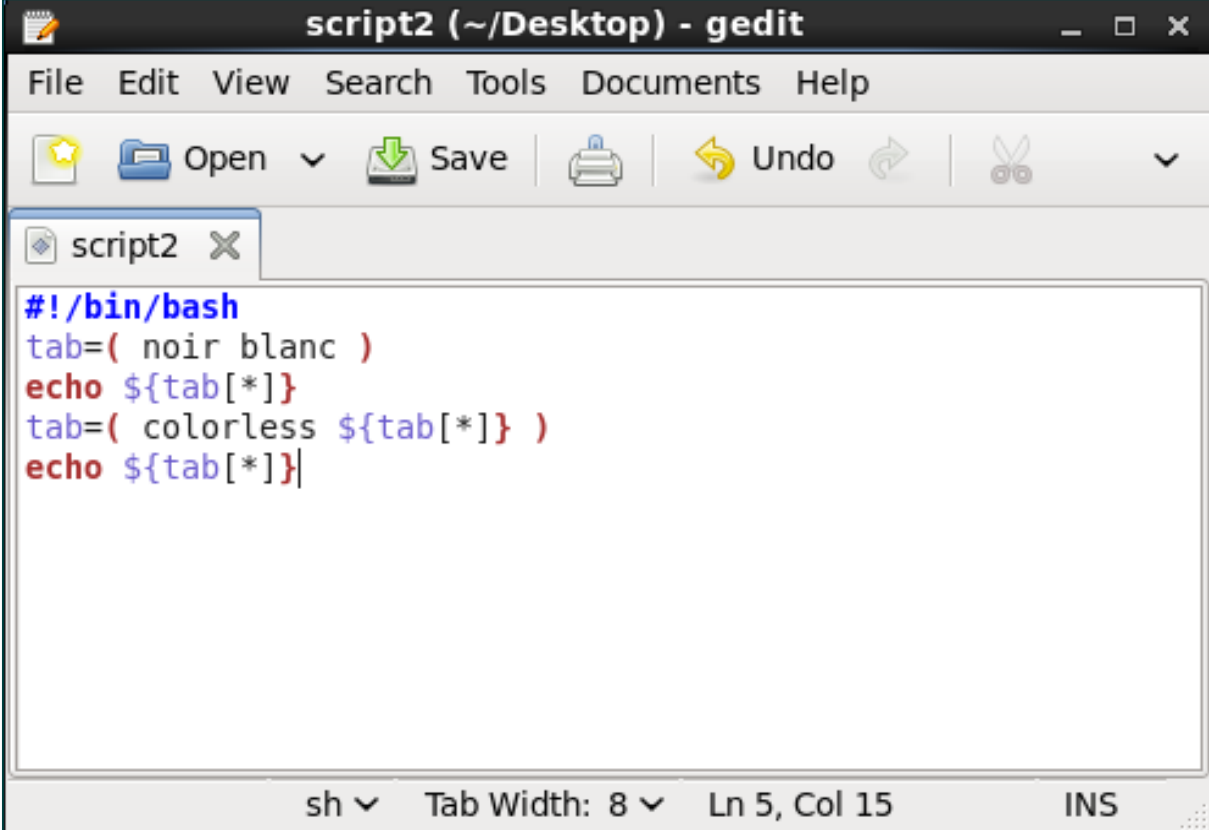


```
root@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
3
bleu jaune rouge couleurs
```

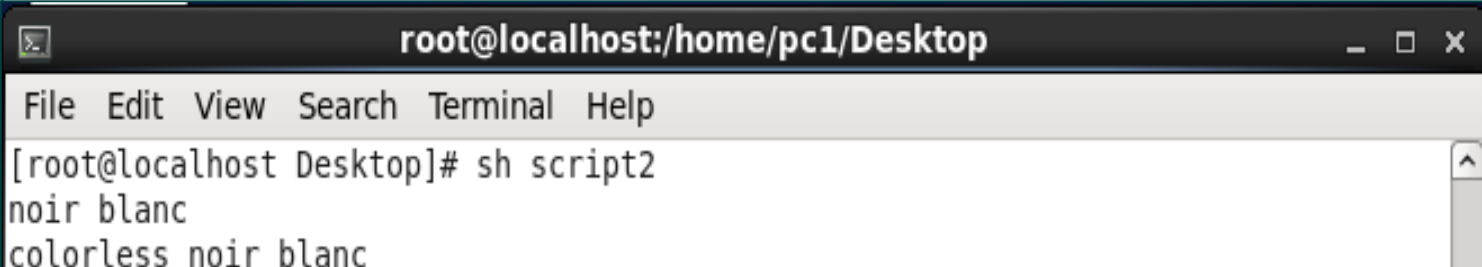
Tableaux

- ❖ 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 :

```
unset nom-tableau
```

- ❖ Supprimer la case d'un tableau :

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

expr : Arithmétique

❖ **expr** : permet d'évaluer des expressions, notamment pour faire des opérations arithmétiques ou des comparaisons de chaînes de caractères .

❖ Les opérateurs arithmétiques sont :

❖ **+** : addition ;

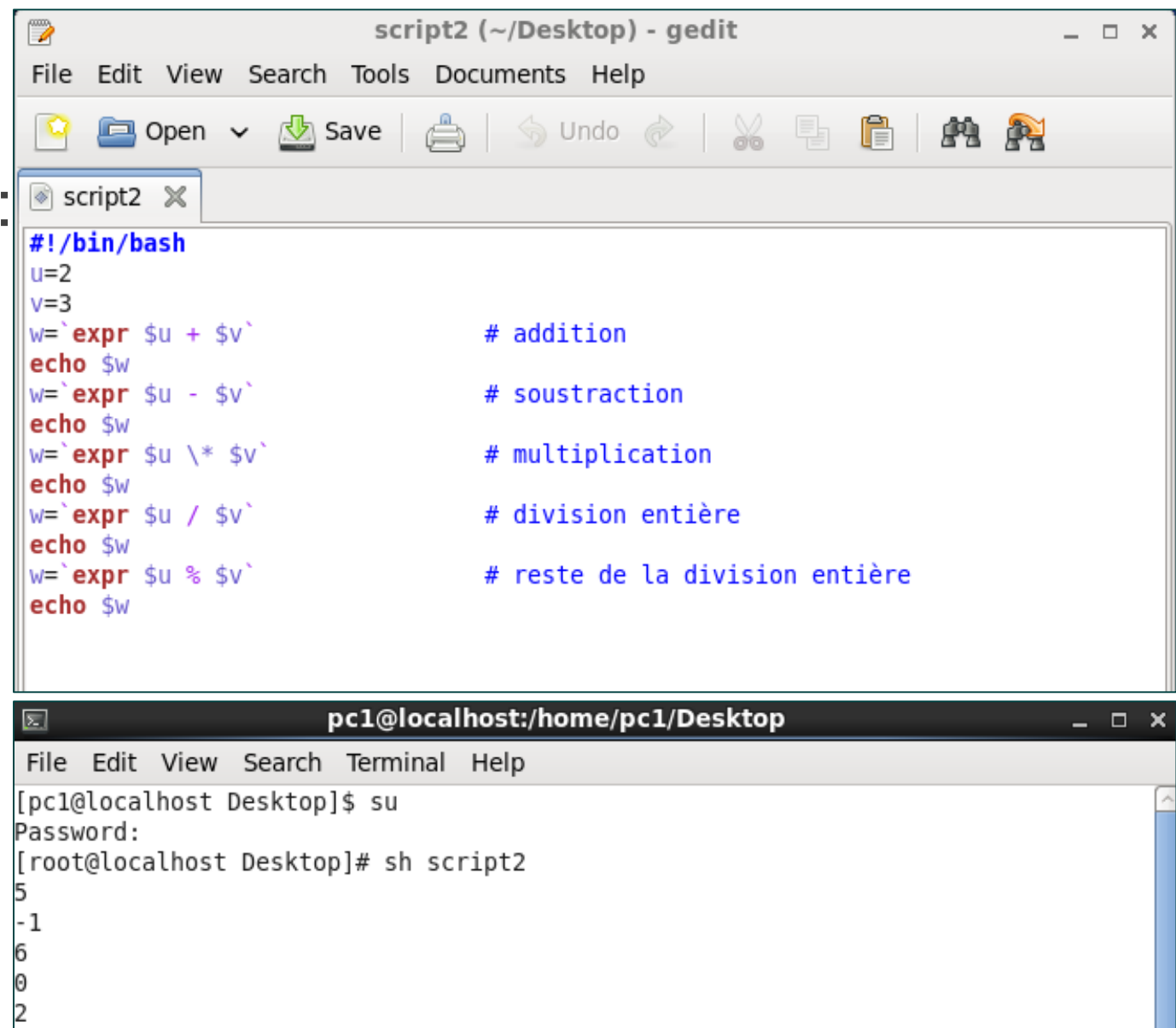
❖ **-** : soustraction ;

❖ ***** : multiplication ;

❖ **/** : division entière ;

❖ **%** : reste de la division ;

❖ **\(** et **\)** : parenthèses.



The image shows two windows. The top window is a Gedit 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`

```
pc1@localhost:/home/pc1/Desktop
File Edit View Search Terminal Help
[root@localhost Desktop]# sh script2
17
17
17
```

```
script2 (~/Desktop) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
script2
#!/bin/bash
chaine=BonjourEsprit2021
echo ${#chaine} # 15
echo `expr length $chaine` # 15
```

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

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

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

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

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.

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

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


expr : manipulation des chaînes

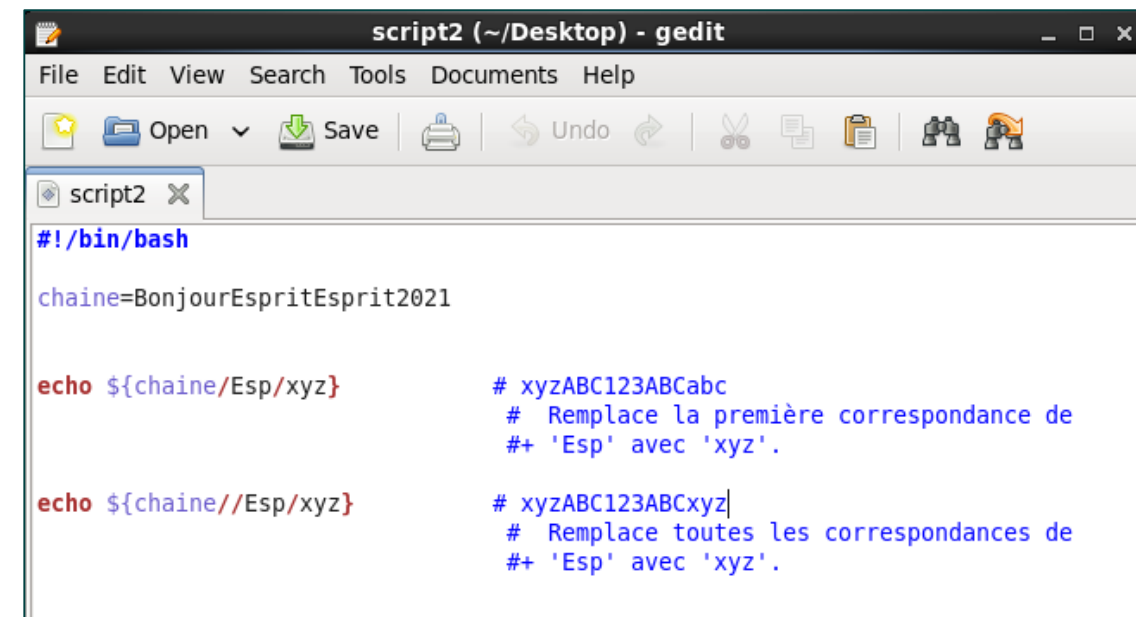
❖ Remplacement de sous-chaîne

➤ **\$ {chaîne/souschaîne/remplacement}**

- Remplace la première correspondance de \$souschaîne par \$remplacement.

➤ **\$ {chaîne//souschaîne/remplacement}**

- Remplace toutes les correspondances de \$souschaîne avec \$remplacement.

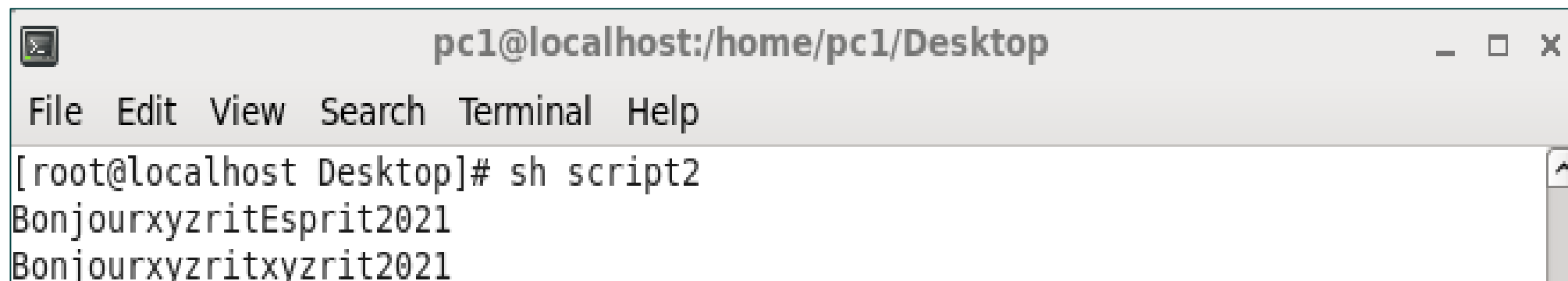


```
#!/bin/bash

chaîne=BonjourEspritEsprit2021

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

echo ${chaîne//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
```

Les structures conditionnelles **test**

La fonction **test** permet d'exprimer une condition ou plusieurs:

- Sur des fichiers
- Sur des nombres
- Sur des chaînes de caractères

Les structures conditionnelles **test** sur les fichiers (1/2)

test -e <nom_fichier> : vrai si l'argument existe

test -f <nom_fichier> : vrai si l'argument est un fichier ordinaire

test -d <nom_fichier> : vrai si l'argument est un répertoire

test -L <nom_fichier> : vrai si l'argument est un lien symbolique

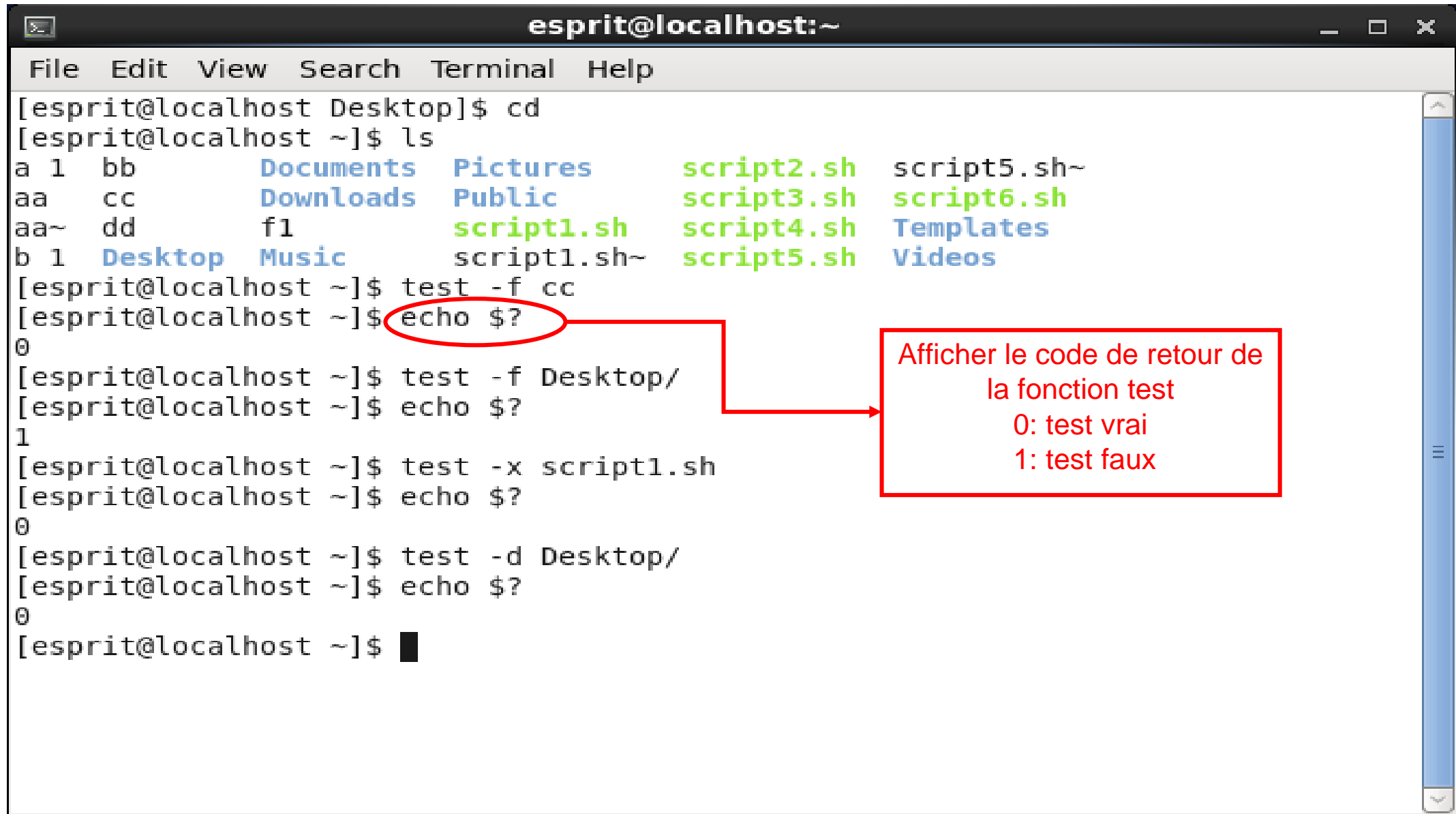
test -r <nom_fichier> : vrai si on a le droit de lire le fichier

test -x <nom_fichier> : vrai si on a le droit d'exécuter le fichier

test -w <nom_fichier> : vrai si on a le droit d'écrire dans le fichier

test -s <nom_fichier> : vrai si l'argument existe et non vide

Les structures conditionnelles **test** sur les fichiers (2/2)



```
esprit@localhost:~  
File Edit View Search Terminal Help  
[esprit@localhost Desktop]$ cd  
[esprit@localhost ~]$ ls  
a 1  bb      Documents Pictures  script2.sh  script5.sh~  
aa  cc      Downloads Public   script3.sh  script6.sh  
aa~ dd      f1      script1.sh  script4.sh  Templates  
b 1  Desktop Music  script1.sh~ script5.sh  Videos  
[esprit@localhost ~]$ test -f cc  
[esprit@localhost ~]$ echo $?  
0  
[esprit@localhost ~]$ test -f Desktop/  
[esprit@localhost ~]$ echo $?  
1  
[esprit@localhost ~]$ test -x script1.sh  
[esprit@localhost ~]$ echo $?  
0  
[esprit@localhost ~]$ test -d Desktop/  
[esprit@localhost ~]$ echo $?  
0  
[esprit@localhost ~]$
```

Afficher le code de retour de la fonction test
0: test vrai
1: test faux

Les structures conditionnelles **test** sur les nombres (1/2)

Pour les expressions arithmétiques, les opérateurs sont:

- eq , -ne, -lt, -le, -gt , -ge

équivalent, différent, inférieur strictement, inférieur ou égale, supérieur strictement, supérieur ou égale

`test "i1" -eq "i2":` Teste si i1 est égal à i2

`test "i1" -ne "i2":` Teste si i1 n'est pas égal à i2

`test "i1" -gt "i2":` Teste si i1 est plus grand que i2

`test "i1" -lt "i2":` Teste si i1 est inférieur à i2

`test "i1" -ge "i2":` Teste si i1 est plus grand ou égal à i2

`test "i1" -le "i2":` Teste si i1 est inférieur ou égal à i2

Les structures conditionnelles **test** sur les nombres (2/2)



```
esprit@localhost:~  
File Edit View Search Terminal Help  
[esprit@localhost Desktop]$ cd  
[esprit@localhost ~]$ v1=50  
[esprit@localhost ~]$ v2=30  
[esprit@localhost ~]$ test $v1 -eq $v2  
[esprit@localhost ~]$ echo $?  
1  
[esprit@localhost ~]$ test $v1 -gt $v2  
[esprit@localhost ~]$ echo $?  
0  
[esprit@localhost ~]$ test $v1 -lt $v2  
[esprit@localhost ~]$ echo $?  
1  
[esprit@localhost ~]$
```

Les structures conditionnelles **test** sur les chaînes de caractères (1/2)

test -n "string" : Teste si la longueur de la chaîne string est différente de zéro

test -z "string": Teste si la chaîne string est égale à zéro

test "s1" **=** "s2": Teste si la chaîne s1 est égale à s2

test "s1" **!=** "s2": Teste si la chaîne s1 n'est pas égale à s2

Les structures conditionnelles **test** sur les chaînes de caractères (2/2)

```
esprit@localhost:~  
File Edit View Search Terminal Help  
[esprit@localhost Desktop]$ cd  
[esprit@localhost ~]$ v1=2A2  
[esprit@localhost ~]$ v2=2A3  
[esprit@localhost ~]$ test $v1 = $v2  
[esprit@localhost ~]$ echo $?  
1  
[esprit@localhost ~]$ test $v1 != $v2  
[esprit@localhost ~]$ echo $?  
0  
[esprit@localhost ~]$ test -n $v1  
[esprit@localhost ~]$ echo $?  
0  
[esprit@localhost ~]$ test -z $v2  
[esprit@localhost ~]$ echo $?  
1  
[esprit@localhost ~]$
```


Les structures conditionnelles **test** vs **[]**

```
test -e fichier  
test $v1 -eq $v2  
test $chaine1 = $chaine2
```



```
[ -e fichier ]  
[ $v1 -eq $v2 ]  
[ $chaine1 = $chaine2 ]
```

Combinaison de conditions

! : Condition Not **[! -d /etc/group]**

-a : Condition AND **[-f script.sh -a -x script.sh]**

-o : Condition OR **[-d script.sh -o -x script.sh]**

\(\) : pour traiter un groupe d'expressions

[-w script.sh -a \(-e fich -o -e fich7 \)]

Effectuer plusieurs tests à la fois

OR = || `test -f file || echo « le fichier file n'existe pas »`

`[-f file] || echo « le fichier file n'existe pas »`

AND = && `test -f file && echo « le fichier file existe »`

Les structures conditionnelles if (1/2)

Syntaxe 1

```
if condition
then
    Traitement 1
else
    Traitement 2
fi
```

Syntaxe 2

```
if condition 1
then
    Traitement 1
else if condition 2
then
    Traitement 2
else if condition 3
then
    traitement 3
fi
fi
fi
```

Syntaxe 3

```
if condition 1
then
    Traitement 1
if condition 2
then
    Traitement 2
else Traitement 3
fi
```

Syntaxe 4

```
if condition ; then
    Traitement
fi
```

Les structures conditionnelles if (2/2)

```
*script1.sh (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
*script1.sh
#!/bin/bash
ls
echo "Choisir un nom de répertoire ou fichier"
read NAME
if [ -f $NAME ]
then echo $NAME ":fichier ordinaire"
elif [ -d $NAME ]
then echo $NAME ":répertoire"
else echo $NAME ":type non traité"
fi
```

```
esprit@localhost:~
File Edit View Search Terminal Help
[esprit@localhost Desktop]$ cd
[esprit@localhost ~]$ gedit script1.sh
[esprit@localhost ~]$ ./script1.sh
a 1 aa~ bb dd Documents fl Pictures script1.sh Templates
aa b 1 cc Desktop Downloads Music Public script1.sh~ Videos
Choisir un nom de répertoire ou fichier
bb
bb :fichier ordinaire
[esprit@localhost ~]$ ./script1.sh
a 1 aa~ bb dd Documents fl Pictures script1.sh Templates
aa b 1 cc Desktop Downloads Music Public script1.sh~ Videos
Choisir un nom de répertoire ou fichier
Downloads
Downloads :répertoire
[esprit@localhost ~]$ ./script1.sh
a 1 aa~ bb dd Documents fl Pictures script1.sh Templates
aa b 1 cc Desktop Downloads Music Public script1.sh~ Videos
Choisir un nom de répertoire ou fichier
music
music :type non traité
[esprit@localhost ~]$
```

Les structures conditionnelles **case** (1/2)

```
case $variable in  
  expr1) commandes ;;  
  expr2) commandes ;;  
  
  .  
  
  .  
  
  .  
  
  *) par_défaut ;;  
esac
```

Les structures conditionnelles case (2/2)

```
script2.sh (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
script2.sh
#!/bin/bash
case $1 in
"Hamdi")
echo "salut Hamdi"
;;
"Mariem")
echo "salut Mariem"
;;
"Manel")
echo "salut Manel"
;;
*)
echo "Je ne te connais pas"
;;
esac
```

```
esprit@localhost:~
File Edit View Search Terminal Help
[esprit@localhost Desktop]$ cd
[esprit@localhost ~]$ gedit script2.sh
[esprit@localhost ~]$ chmod u+x script2.sh
[esprit@localhost ~]$ ./script2.sh Mariem
salut Mariem
[esprit@localhost ~]$ ./script2.sh Hamdi
salut Hamdi
[esprit@localhost ~]$ ./script2.sh
Je ne te connais pas
[esprit@localhost ~]$ ./script2.sh manel
Je ne te connais pas
[esprit@localhost ~]$
```

Les structures itératives **for** (1/3)

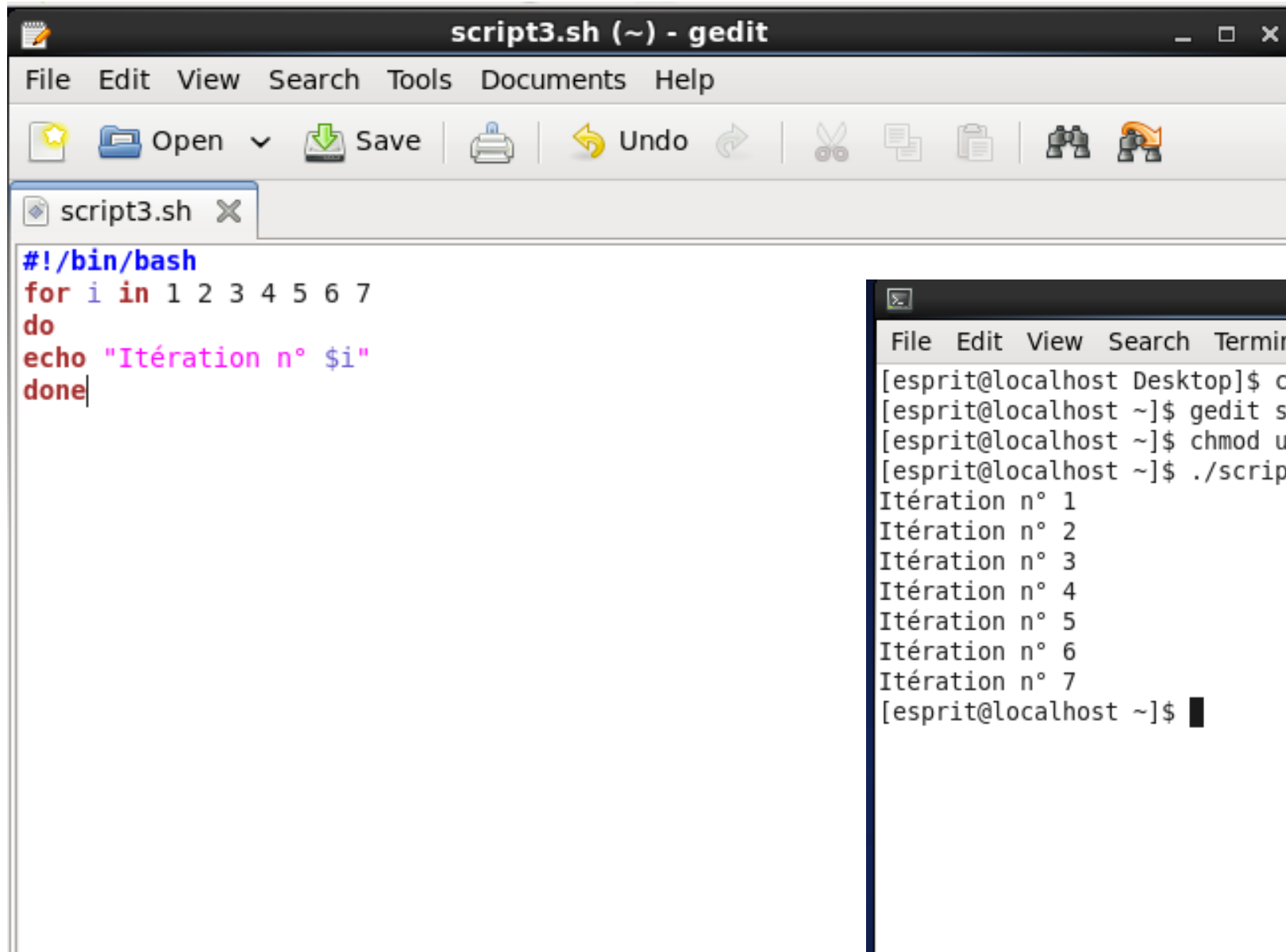
for variable [**in** liste]

do

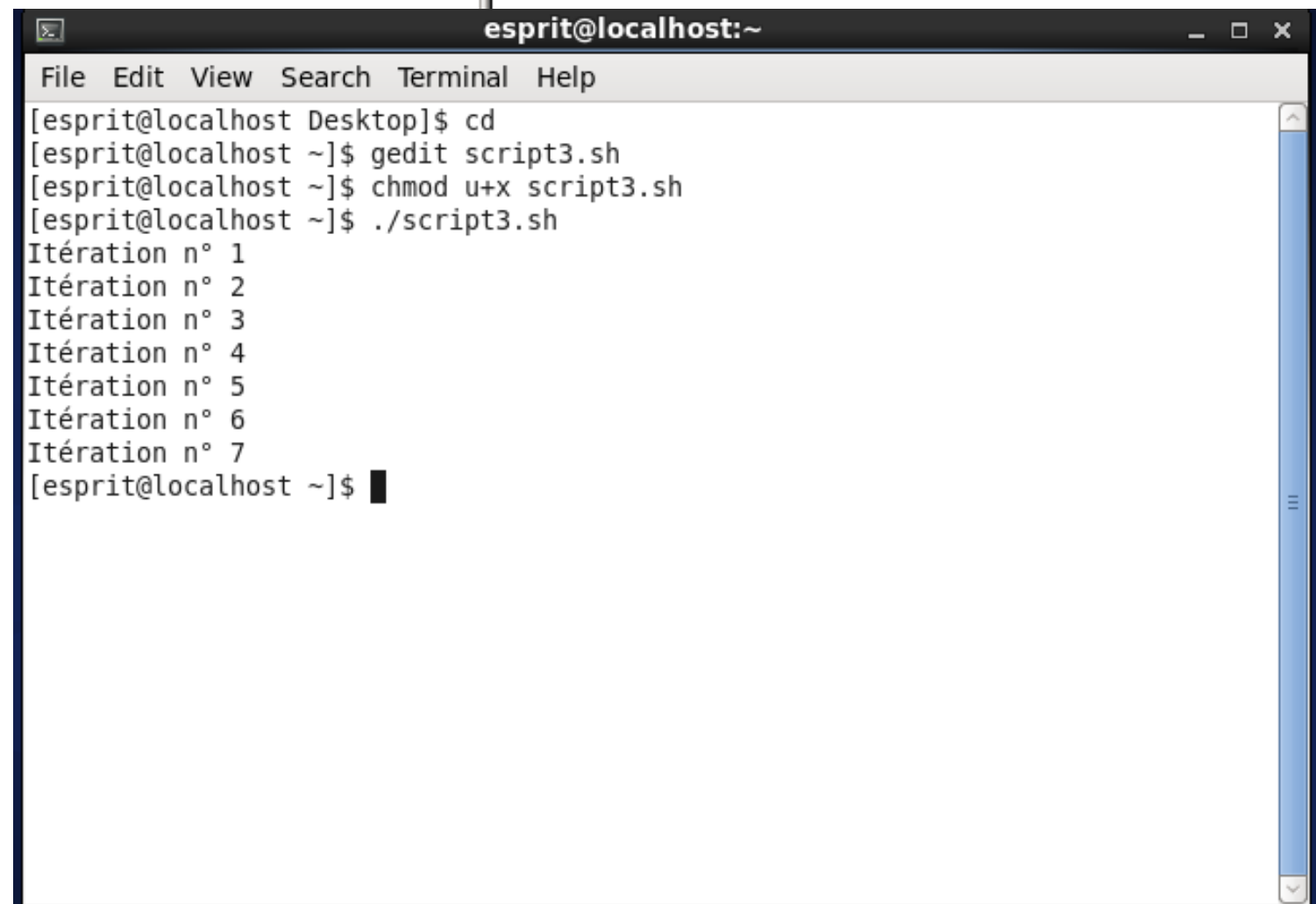
commandes (*utilisant \$variable*)

done

Les structures itératives for (2/3)



```
script3.sh (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
script3.sh
#!/bin/bash
for i in 1 2 3 4 5 6 7
do
echo "Itération n° $i"
done
```



```
esprit@localhost:~
File Edit View Search Terminal Help
[esprit@localhost Desktop]$ cd
[esprit@localhost ~]$ gedit script3.sh
[esprit@localhost ~]$ chmod u+x script3.sh
[esprit@localhost ~]$ ./script3.sh
Itération n° 1
Itération n° 2
Itération n° 3
Itération n° 4
Itération n° 5
Itération n° 6
Itération n° 7
[esprit@localhost ~]$
```

Les structures itératives for (3/3)

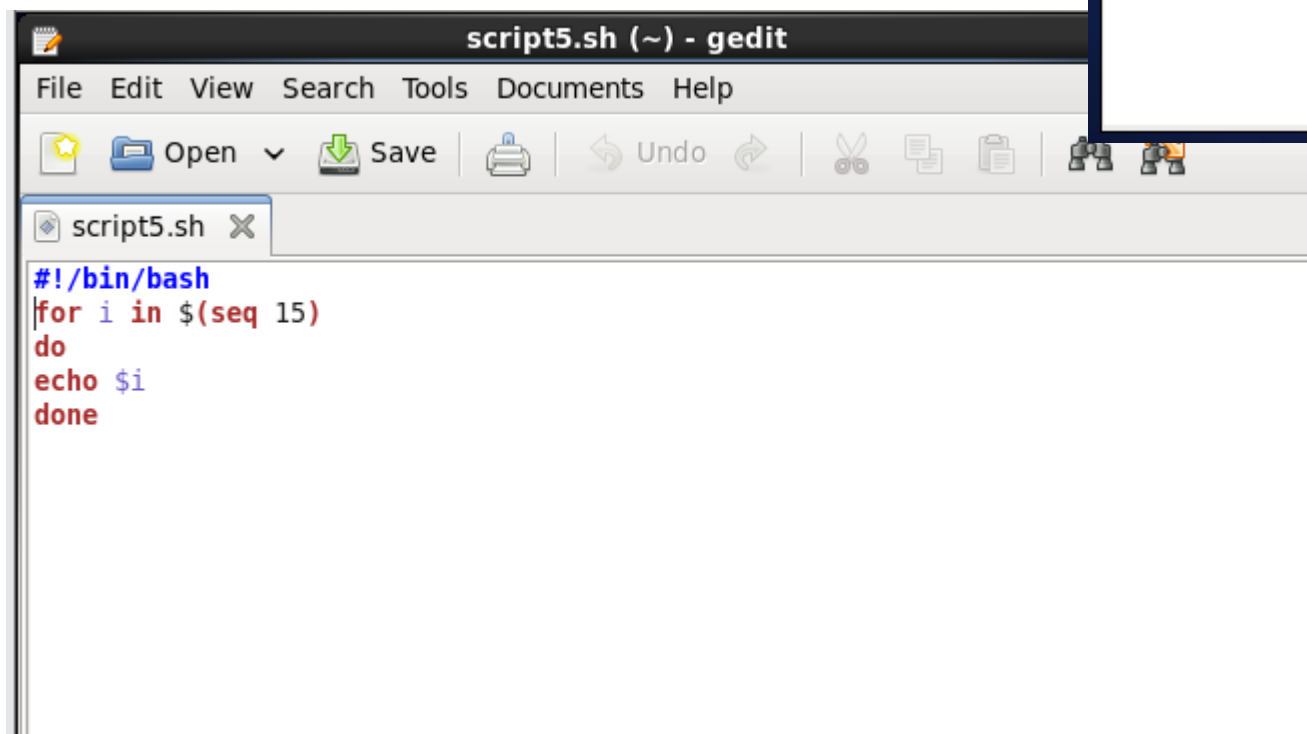
```
script4.sh (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
script4.sh
#!/bin/bash
for fichier in `ls /home/esprit/*`
do
if [ -r $fichier ]
then
echo "$fichier est accessible en lecture"
fi
done
```

```
esprit@localhost:~
File Edit View Search Terminal Help
[esprit@localhost Desktop]$ cd
[esprit@localhost ~]$ gedit script4.sh
[esprit@localhost ~]$ chmod u+x script4.sh
[esprit@localhost ~]$ ./script4.sh
/home/esprit/aa est accessible en lecture
/home/esprit/aa~ est accessible en lecture
/home/esprit/bb est accessible en lecture
/home/esprit/cc est accessible en lecture
/home/esprit/dd est accessible en lecture
/home/esprit/fl est accessible en lecture
/home/esprit/script1.sh est accessible en lecture
/home/esprit/script1.sh~ est accessible en lecture
/home/esprit/script2.sh est accessible en lecture
/home/esprit/script3.sh est accessible en lecture
/home/esprit/script4.sh est accessible en lecture
[esprit@localhost ~]$
```

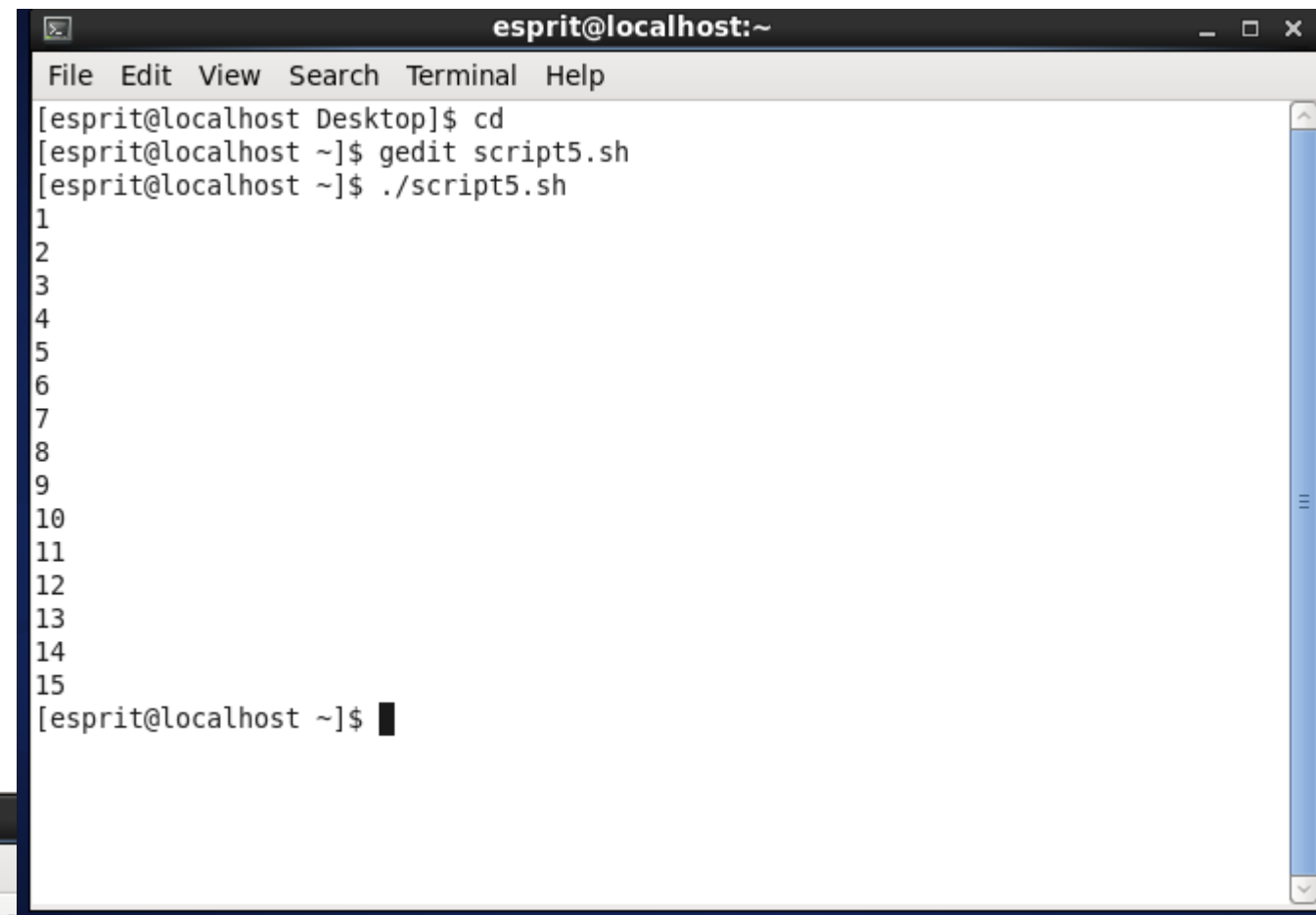
Commande seq (1/2)

```
for i in 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
do
echo $i
done
```

Très pénible ! Pour itérer par exemple 250 fois, vous devez saisir tout cela au clavier ! Heureusement, **il y a un « raccourci »**, la **commande seq**, qui affiche une séquence de nombres allant de 1 jusqu'au maximum indiqué

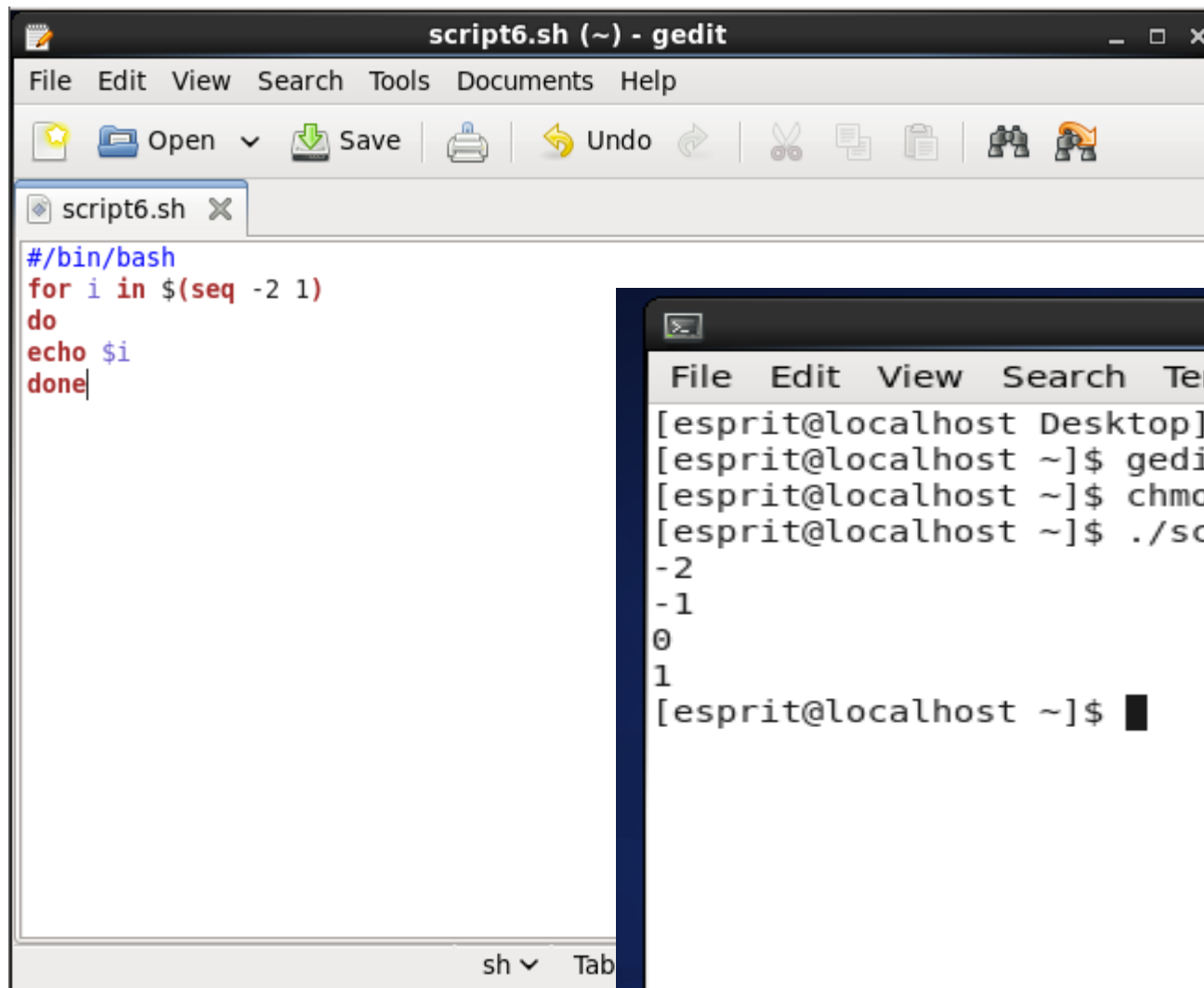
A screenshot of a gedit window titled "script5.sh (~) - gedit". The window has a menu bar with "File", "Edit", "View", "Search", "Tools", "Documents", and "Help". Below the menu bar is a toolbar with icons for "Open", "Save", "Undo", and "Redo". The main text area shows the following code:

```
#!/bin/bash
for i in $(seq 15)
do
echo $i
done
```

A screenshot of a terminal window titled "esprit@localhost:~". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the following commands and output:

```
[esprit@localhost Desktop]$ cd
[esprit@localhost ~]$ gedit script5.sh
[esprit@localhost ~]$ ./script5.sh
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
[esprit@localhost ~]$
```

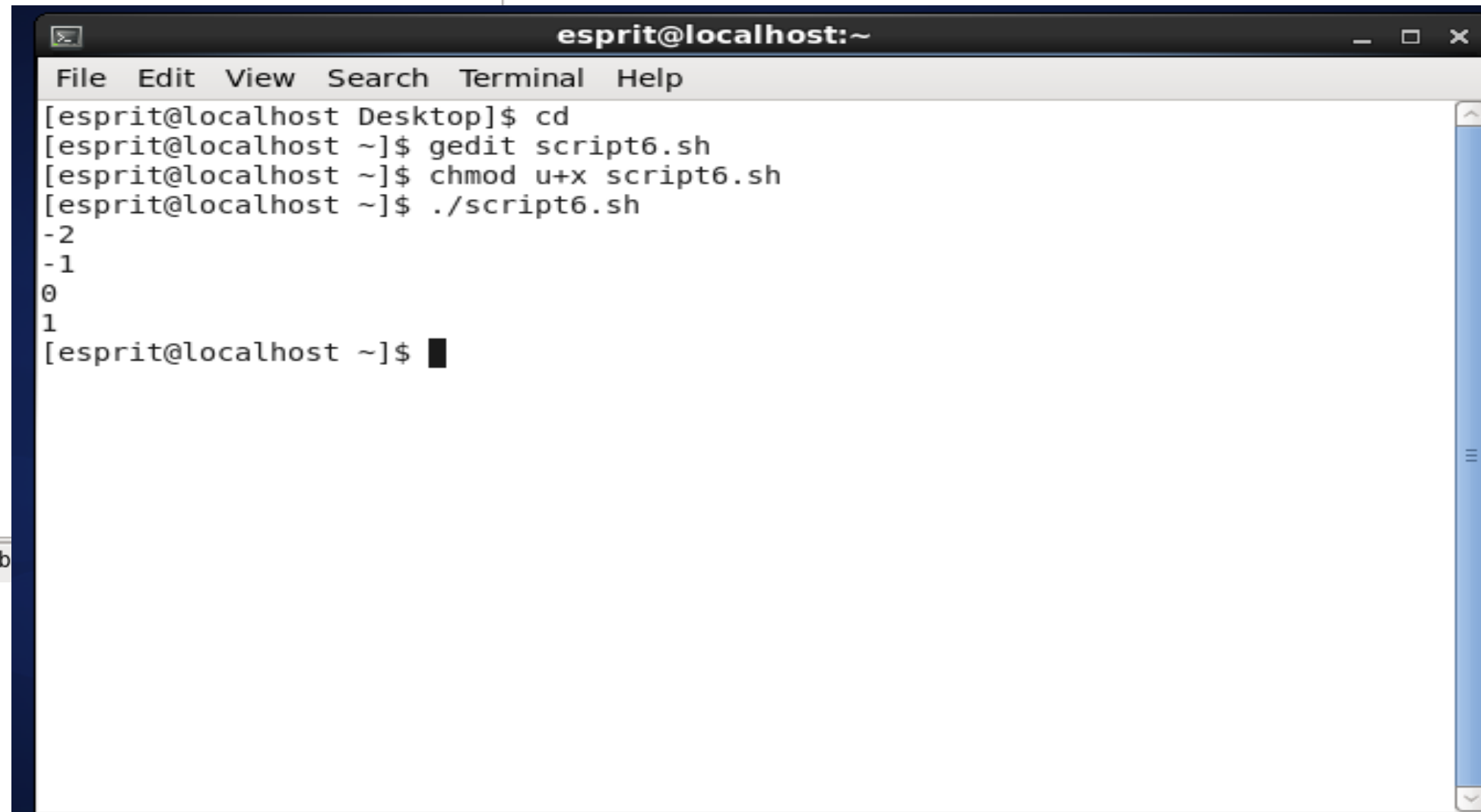
Commande seq (2/2)



A screenshot of a gedit editor window titled "script6.sh (~) - gedit". The window has a menu bar with "File", "Edit", "View", "Search", "Tools", "Documents", and "Help". Below the menu bar is a toolbar with icons for "Open", "Save", "Undo", and "Redo". The script content is as follows:

```
#!/bin/bash
for i in $(seq -2 1)
do
echo $i
done
```

At the bottom of the window, there is a status bar showing "sh" and "Tab".



A screenshot of a terminal window titled "esprit@localhost:~". The window has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the following commands and output:

```
[esprit@localhost Desktop]$ cd
[esprit@localhost ~]$ gedit script6.sh
[esprit@localhost ~]$ chmod u+x script6.sh
[esprit@localhost ~]$ ./script6.sh
-2
-1
0
1
[esprit@localhost ~]$
```

Les structures itératives **while**, **until**

while commande 1

do

commandes

done

La boucle **while** permet d'exécuter les commandes présentes entre le **do** et le **done** tant que la commande1 placée à droite du **while** retourne un code vrai.

until commande1

do

commandes

done

la commande **until** exécute les commandes situées entre le **do** et le **done** tant que la commande1 située à droite du **until** retourne un code faux.

Les fonctions

- ❑ Une fonction est un bloc de code qui implémente un ensemble d'opérations.

```
function nom_fonction {
```

```
  commande...
```

```
}
```

Ou

```
nom_fonction () {
```

```
  commande...
```

```
}
```

- ❖ **Les fonctions sont appelées, *lancées*, simplement en invoquant leur noms.**

Fonctions

- ❖ Les fonctions peuvent récupérer des arguments qui leur sont passés et renvoyer un **code de sortie** au script pour utilisation ultérieure.

nom_fonction \$arg1 \$arg2

- ❖ La fonction se réfère aux arguments passés par leur position (comme s'ils étaient des **paramètres positionnels**), c'est-à-dire \$1, \$2 et ainsi de suite.

Fonction avec des paramètres

```
#!/bin/bash
usage() {
    echo "Usage: $0 filename"
    exit 1
}
file_exists() {
    test -f "$1" && return 0 || return 1
}
test $# -eq 0 && usage
if file_exists "$1"
then
    echo "File found"
else
    echo "File not found"
fi
```

Code de retour d'une fonction

- ❖ **code de retour:** Les fonctions renvoient une valeur, appelée un *code* (ou *état*) de retour .
- ❖ Le code de retour peut être explicitement spécifié par une instruction **return**, sinon, il s'agit du code de retour de la dernière commande de la fonction (0 en cas de succès et une valeur non nulle comprise entre 1 et 255).
- ❖ le code retourné par une fonction est récupérable grâce à la variable \$?.

Terminaisons

- ❖ Un script : **exit n** (n = code de retour indiquant succès ou échec).
- ❖ Une fonction : **return n** (n = code de retour indiquant succès ou échec).
- ❖ Une boucle : **break**

Exemple :

```
#!/bin/bash
E_PARAM_ERR=250
EGAL=251
max ()
{
if [ $# -lt 2]
then
    return $E_PARAM_ERR
fi
```

```
if [ "$1" -eq "$2" ]
then
    return $EGAL
else
    if [ "$1" -gt "$2" ]
    then
        return $1
    else
        return $2
    fi fi }
```

```
# Si moins de deux paramètres passés à la fonction.
# Code de retour si les deux paramètres sont égaux.
# Envoie le plus important des deux entiers.
```

Exemple (suite)

Max 33 34 **#appel de la fonction.**

return_val=\$?

if ["\$return_val" -eq \$E_PARAM_ERR]

then

 echo "Vous devez donner deux arguments à la fonction."

elif ["\$return_val" -eq \$EGAL]

then

 echo "Les deux nombres sont identiques."

else

 echo "Le plus grand des deux nombres est \$return_val."

fi

exit 0