

# Les bases du shell

---

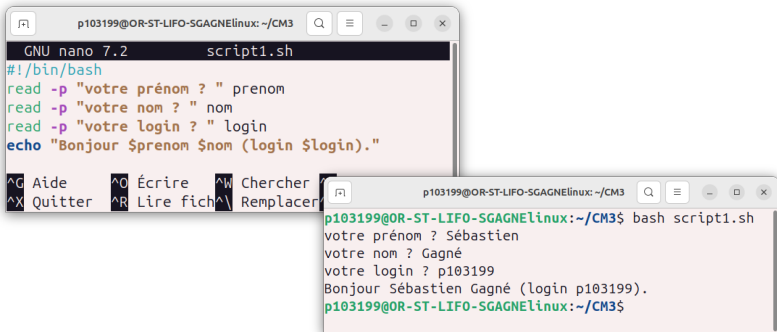
Sébastien GAGNÉ, Université d'Orléans

L1 Pratique du système UNIX — S2

# Premiers scripts shell

Un **script shell** est un **fichier texte** contenant une suite de **commandes UNIX** qui s'exécutent l'une après l'autre.

Il permet l'automatisation des tâches, résumées en un seul appel, celui du script.



The image displays two terminal windows from a Linux environment. The top window shows the creation of a shell script using the nano text editor. The script content is as follows:

```
GNU nano 7.2 script1.sh
#!/bin/bash
read -p "votre prénom ? " prenom
read -p "votre nom ? " nom
read -p "votre login ? " login
echo "Bonjour $prenom $nom (login $login)."
```

The bottom window shows the execution of the script. The user runs `bash script1.sh`, and the script prompts for a first name, last name, and login, then outputs a personalized greeting.

```
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$ bash script1.sh
votre prénom ? Sébastien
votre nom ? Gagné
votre login ? p103199
Bonjour Sébastien Gagné (login p103199).
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$
```

# Détails sur les fichiers script

---

La première ligne du script commence par #!

Le "**shebang**", représenté par #!, est un en-tête de fichier qui indique au système d'exploitation que ce fichier est un **script** (ensemble de commandes sous forme textuelle) et non **pas un fichier binaire** (ensemble d'instructions en code "machine").

Sur la même ligne est précisé le **shell** interpréteur permettant d'exécuter ce script.

Exemple :      `#!/bin/bash`

# Détails sur les fichiers script

---

Même si cela n'a rien d'obligatoire, il est habituel de nommer le fichier script avec l'extension ".sh"

Les lignes écrites dans le script sont composées

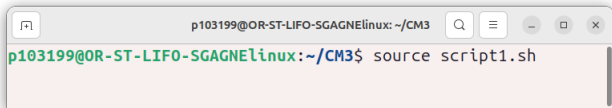
- de commandes UNIX élémentaires,
- de variables,
- de boucles répétitives,
- de structures conditionnelles,
- de fonctions éventuellement.

# Exécution d'un fichier script

---

On peut exécuter le script de plusieurs façons :

1. par la commande `source monScript.sh`  
ou son alias : `. monScript.sh`

A screenshot of a Linux terminal window. The title bar shows the user 'p103199' and the host 'OR-ST-LIFO-SGAGNElinux' with the current directory '~/CM3'. The terminal prompt is 'p103199@OR-ST-LIFO-SGAGNElinux:~/CM3\$'. The user has entered the command 'source script1.sh'.

```
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ source script1.sh
```

Cette méthode lit une à une les commandes du script et les exécute, comme si elles avaient été écrites dans le terminal.

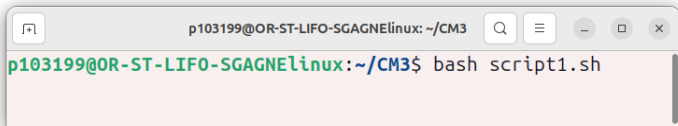
Ainsi, toute variable créée ou modifiée dans le script l'est encore après l'exécution.

# Exécution d'un fichier script

---

On peut exécuter le script de plusieurs façons :

2. par la commande `bash monScript.sh`



Cette méthode crée un processus fils `bash` dans lequel le script s'exécute.

Ainsi, toute variable créée ou modifiée dans le script ne l'est plus après l'exécution, puisqu'elle était liée au processus `bash` fils.

# Exécution d'un fichier script

---

On peut exécuter le script de plusieurs façons :

3. par la commande `./monScript.sh`

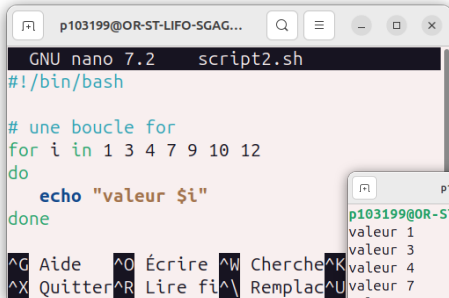


(nécessite les droits d'exécution sur le fichier)

Un processus `monScript.sh` est créé dans le terminal, et toute variable créée ou modifiée dans le script ne l'est plus après l'exécution, étant liée au processus `monScript.sh` fils.

# La boucle **for**

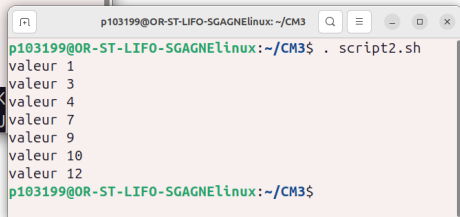
La boucle `for` permet de répéter une commande en faisant parcourir un ensemble de valeurs (parfois calculées avec une expansion) à une variable.



```
GNU nano 7.2  script2.sh
#!/bin/bash

# une boucle for
for i in 1 3 4 7 9 10 12
do
    echo "valeur $i"
done
```

^G Aide ^O Écrire ^W Cherche ^K  
^X Quitter ^R Lire fi ^\ Remplac ^U

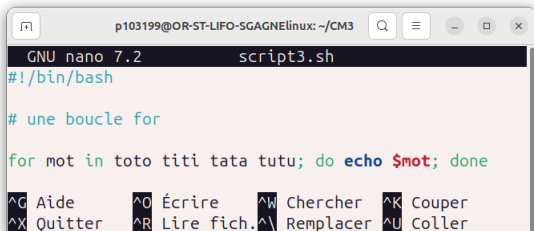


```
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$ . script2.sh
valeur 1
valeur 3
valeur 4
valeur 7
valeur 9
valeur 10
valeur 12
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$
```



# La boucle **for**

La syntaxe de la boucle `for` peut être "en ligne", en utilisant l'opérateur `;` :

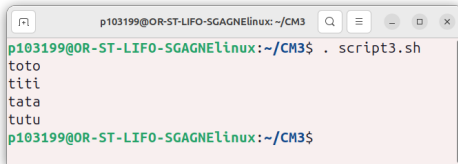


```
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3
GNU nano 7.2      script3.sh
#!/bin/bash

# une boucle for

for mot in toto titi tata tutu; do echo $mot; done

^G Aide      ^O Écrire    ^W Chercher   ^K Couper
^X Quitter    ^R Lire fich.^_ Remplacer   ^U Coller
```



```
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ . script3.sh
toto
titi
tata
tutu
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$
```

# Boucle **for** et commande **seq**

La commande `seq` génère des suites arithmétiques.

seq 5 1 2 3 4 5

seq 2 7                      2 3 4 5 6 7

seq 7 6 28                      7 13 19 25

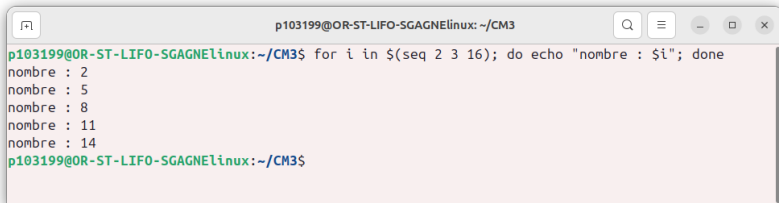
```
seq 17 -4 5          17 13 9 5
```

```
seq a r b
```

a a+r ... a+kr ...  
(sans dépasser b)

# Boucle **for** et commande **seq**

Par expansion de la commande `seq`, on peut créer une boucle `for` où les valeurs sont prises dans une suite arithmétique.

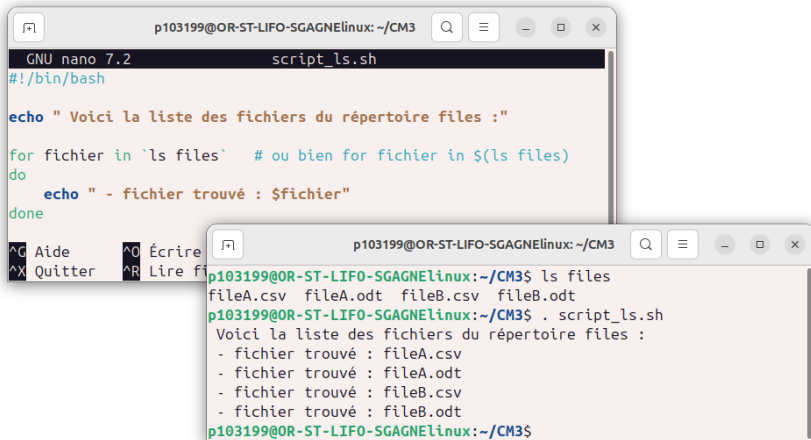
A terminal window with a title bar showing the user 'p103199' and the directory '~/CM3'. The prompt is 'p103199@OR-ST-LIFO-SGAGNElinux:~/CM3\$'. The command entered is 'for i in \$(seq 2 3 16); do echo "nombre : \$i"; done'. The output shows five lines: 'nombre : 2', 'nombre : 5', 'nombre : 8', 'nombre : 11', and 'nombre : 14'. The prompt returns to 'p103199@OR-ST-LIFO-SGAGNElinux:~/CM3\$'.

```
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ for i in $(seq 2 3 16); do echo "nombre : $i"; done
nombre : 2
nombre : 5
nombre : 8
nombre : 11
nombre : 14
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$
```

La commande `seq 2 3 16` de cet exemple *one-line* est interprétée par l'expansion, ce qui fait tourner la boucle `for` sur les valeurs 2, 5, 8, 11 et 14.

# Boucle **for** et autres commandes

On peut aussi utiliser une expansion de la commande `ls` pour alimenter les valeurs de la boucle `for` :



The image shows two terminal windows. The top window is a nano editor editing a file named `script_ls.sh`. The script contains the following code:

```
#!/bin/bash

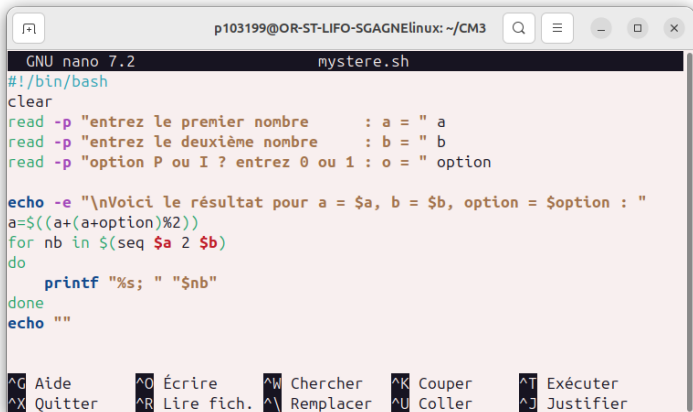
echo " Voici la liste des fichiers du répertoire files :"

for fichier in `ls files`  # ou bien for fichier in $(ls files)
do
    echo " - fichier trouvé : $fichier"
done
```

The bottom window shows the execution of the script. First, the user runs `ls files` to list the contents of the `files` directory. Then, they run `. script_ls.sh` to execute the script, which produces the following output:

```
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$ ls files
fileA.csv fileA.odt fileB.csv fileB.odt
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$ . script_ls.sh
Voici la liste des fichiers du répertoire files :
- fichier trouvé : fileA.csv
- fichier trouvé : fileA.odt
- fichier trouvé : fileB.csv
- fichier trouvé : fileB.odt
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3$
```

# Analyse de code shell



The screenshot shows a terminal window with the title bar "p103199@OR-ST-LIFO-SGAGNELinux: ~/CM3". The window contains a nano 7.2 editor editing a file named "mystere.sh". The script's content is as follows:

```
#!/bin/bash
clear
read -p "entrez le premier nombre      : a = " a
read -p "entrez le deuxième nombre     : b = " b
read -p "option P ou I ? entrez 0 ou 1 : o = " option

echo -e "\nVoici le résultat pour a = $a, b = $b, option = $option : "
a=$((a+(a+option)%2))
for nb in $(seq $a 2 $b)
do
    printf "%s; " "$nb"
done
echo ""
```

At the bottom of the terminal, there is a status bar with keyboard shortcuts for various nano editor functions:

|                   |                      |                      |                  |                     |
|-------------------|----------------------|----------------------|------------------|---------------------|
| <b>^G</b> Aide    | <b>^O</b> Écrire     | <b>^W</b> Chercher   | <b>^K</b> Couper | <b>^T</b> Exécuter  |
| <b>^X</b> Quitter | <b>^R</b> Lire fich. | <b>^_\</b> Remplacer | <b>^U</b> Coller | <b>^J</b> Justifier |

Que fait concrètement ce script `mystere.sh` ?

# Instruction conditionnelle **if**

---

La conditionnelle **if** est utilisée dans un script shell pour exécuter des commandes ou des blocs de code en fonction de certaines conditions.

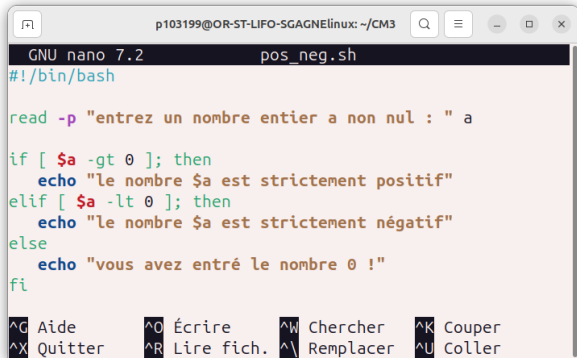
Syntaxe générale :

`if [ condition ]`    ou    `if test condition`

```
if [ condition_1 ]; then
    # bloc de commandes à exécuter
    # si condition_1 est vraie
elif [ condition_2 ]; then
    # sinon, bloc de commandes à exécuter
    # si condition_2 est vraie
else
    # bloc de commandes à exécuter
    # si aucune condition n'est vraie
fi
```

# Instruction conditionnelle **if**

Un exemple avec quelques tests :



```
GNU nano 7.2 pos_neg.sh
#!/bin/bash

read -p "entrez un nombre entier a non nul : " a

if [ $a -gt 0 ]; then
    echo "le nombre $a est strictement positif"
elif [ $a -lt 0 ]; then
    echo "le nombre $a est strictement négatif"
else
    echo "vous avez entré le nombre 0 !"
fi

^G Aide      ^O Écrire    ^W Chercher  ^K Couper
^X Quitter   ^R Lire fich.^_ Remplacer  ^U Coller
```

Attention, les crochets qui délimitent les conditions doivent être entourés d'espaces.

# Comparaisons d'entiers

---

[ \$a -lt \$b ]

vrai si  $a < b$

[ \$a -gt \$b ]

vrai si  $a > b$

[ \$a -le \$b ]

vrai si  $a \leq b$

[ \$a -ge \$b ]

vrai si  $a \geq b$

[ \$a -eq \$b ]

vrai si  $a = b$

[ \$a -ne \$b ]

vrai si  $a \neq b$



# Comparaisons de chaînes

---

[ \$x ]                      vrai si la chaîne est non vide

[ -z \$x ]                      vrai si la chaîne est vide

[ \$x = \$y ]                      vrai si x et y sont identiques

[ \$x == \$y ]                      vrai si x et y sont identiques

[ \$x != \$y ]                      vrai si x et y sont différentes

[[ \$x < \$y ]]                      vrai si  $x < y$  (ordre alphabétique)

[[ \$x > \$y ]]                      vrai si  $x > y$  (ordre alphabétique)

# Conditions sur les fichiers

---

|                          |   |
|--------------------------|---|
| <code>[ -e \$f ]</code>  | vrai si <code>f</code> est un fichier existant      |
| <code>[ -f \$f ]</code>  | vrai si <code>f</code> est un fichier classique     |
| <code>[ -s \$f ]</code>  | vrai si <code>f</code> est un fichier non vide      |
| <code>[ -d \$f ]</code>  | vrai si <code>f</code> est un répertoire            |
| <code>[ -r \$f ]</code>  | vrai si <code>f</code> est un fichier lisible       |
| <code>[ -w \$f ]</code>  | vrai si <code>f</code> est un fichier modifiable    |
| <code>[ -x fich ]</code> | vrai si <code>fich</code> est un fichier exécutable |

# Opérateurs logiques

[ ! c ]

**vrai** ssi c est **faux**

[ c1 -a c2 ]

**vrai** ssi c1 et c2 sont **vrais**

[ c1 -o c2 ]

**vrai** ssi c1 ou c2 sont **vrais**



```
p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3
GNU nano 7.2                                operateurs.sh
#!/bin/bash
echo "entrez un nombre a qui doit être :"
echo "a) inférieur à 100 ou supérieur à 200,"
echo "b) multiple de 3 et multiple de 5,"
echo "c) non multiple de 7."
read -p "a = " a
if [ $a -lt 100 -o $a -gt 200 ]; then echo "a) respectée"; fi
if [ $((a%3)) -eq 0 -a $((a%5)) -eq 0 ]; then echo "b) respectée."; fi
if [ ! $((a%7)) -eq 0 ]; then echo "c) respectée."; fi

^G Aide      ^O Écrire    ^W Chercher  ^K Couper    ^T Exécuter
^X Quitter   ^R Lire fich.^_ Remplacer  ^U Coller    ^J Justifier
```

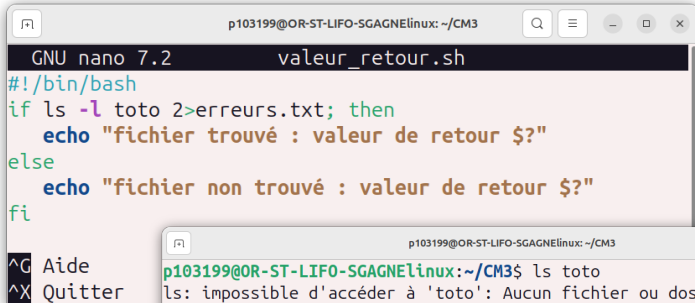
# Valeur de retour d'un processus

---

- Un processus exécuté se termine en retournant une **valeur de retour**.
- Cette valeur est 0 si le processus s'est exécuté sans erreur. Sinon c'est une valeur non nulle.
- Dans une conditionnelle **if**, on peut utiliser une valeur de retour de processus comme condition.
- Lors d'une exécution de script, la variable **?**, dont la valeur est accessible par l'expansion **\$?**, donne la valeur de retour du dernier processus exécuté.

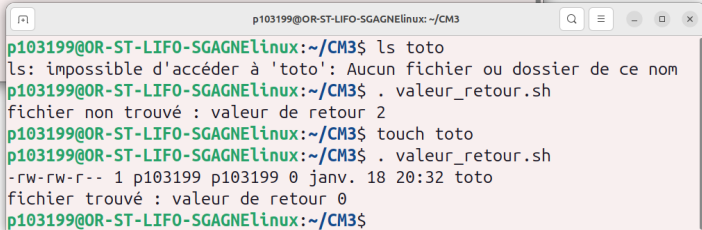
# Valeur de retour d'un processus

## Exemple



A screenshot of a terminal window running GNU nano 7.2. The file being edited is 'valeur\_retour.sh'. The script contains a conditional statement that checks if a file named 'toto' exists. If it does, it prints 'fichier trouvé : valeur de retour \$?'. If it doesn't, it prints 'fichier non trouvé : valeur de retour \$?'. The window title is 'p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3'.

```
GNU nano 7.2          valeur_retour.sh
#!/bin/bash
if ls -l toto 2>erreurs.txt; then
    echo "fichier trouvé : valeur de retour $?"
else
    echo "fichier non trouvé : valeur de retour $?"
fi
^G Aide
^X Quitter
```



A screenshot of a terminal window showing the execution of the script 'valeur\_retour.sh'. The user first runs 'ls toto', which returns an error because the file does not exist. Then, the user runs './valeur\_retour.sh', which prints 'fichier non trouvé : valeur de retour 2'. Next, the user runs 'touch toto' to create the file. Finally, the user runs './valeur\_retour.sh' again, which prints 'fichier trouvé : valeur de retour 0'. The window title is 'p103199@OR-ST-LIFO-SGAGNElinux: ~/CM3'.

```
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ ls toto
ls: impossible d'accéder à 'toto': Aucun fichier ou dossier de ce nom
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ . valeur_retour.sh
fichier non trouvé : valeur de retour 2
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ touch toto
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$ . valeur_retour.sh
-rw-rw-r-- 1 p103199 p103199 0 janv. 18 20:32 toto
fichier trouvé : valeur de retour 0
p103199@OR-ST-LIFO-SGAGNElinux:~/CM3$
```