



Linux

# Introduction aux scripts shells

# Définition

---

- Un script shell est un fichier permettant d'automatiser une série d'opérations sur un ordinateur / serveur.
- Il se présente sous la forme d'un fichier contenant une ou plusieurs commandes qui seront exécutées de manière séquentielle.

# Exécuter des commandes

---

- Deux possibilités :
  - ☐ Taper dans un shell (terminal) toutes les commandes ➔ pratique de base mais très long quand il y a beaucoup de commandes.
  - ☐ Rassembler dans un fichier script toutes les instructions.

# Créer un fichier de script

---

- Utilisez la commande « nano » suivi du nom que vous voulez attribuer à votre fichier de script.
- Exemple : `(sudo) nano test.sh`
- `.sh` est l'extension principale des fichiers scripts sur Linux
- L'utilisation de `sudo` (superutilisateur) est souvent nécessaire pour pouvoir enregistrer votre fichier (cela va dépendre de l'endroit où vous voulez sauvegarder.)

# Commandes tapées normalement

---

```
echo Mon premier script
```

```
echo Liste des fichiers :
```

```
ls -la
```

# Rassembler les commandes dans un script

Fichier Édition Affichage Rechercher Terminal Aide

GNU nano 2.9.3

```
#!/bin/bash
# Indique au système que l'argument qui suit est le programme utilisé pour
# exécuter le fichier
# Les "#" servent à mettre en commentaire le texte qui suit comme ici
echo Mon premier script
echo Liste des fichiers :
ls -la
```

# Comment faire fonctionner un script ?

---

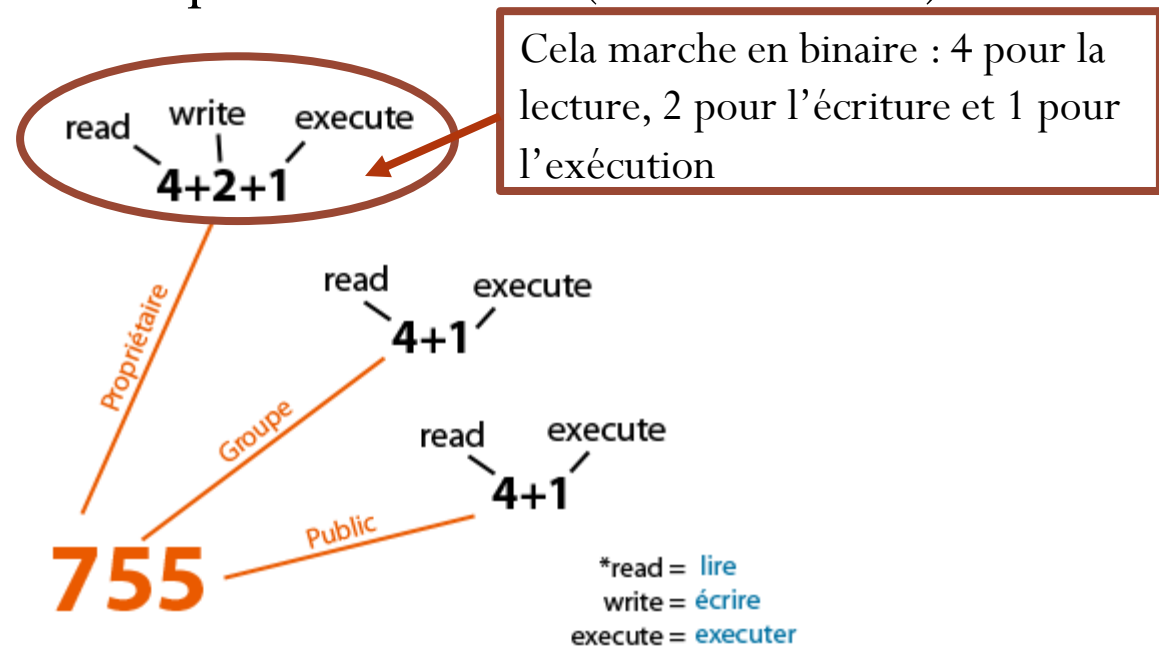
- Il faut simplement le rendre exécutable :

☐ Commande « chmod »

- « chmod » sert à changer les permissions de lecture (R), d'écriture (W) et d'exécution (X) d'un fichier ou dossier, soit en mettant plus de droits sur la cible indiquée, soit en enlevant ces droits.

# 3 groupes concernés par la commande

- 3 groupes de droits (Propriétaire, Groupe, User) pour 3 actions différentes pour un fichier (ici « test.sh ») :



On a donc la commande « `chmod 755 test.sh` » à effectuer pour attribuer les droits RWX au propriétaire, RX au groupe et RX à l'utilisateur



# Autre méthode

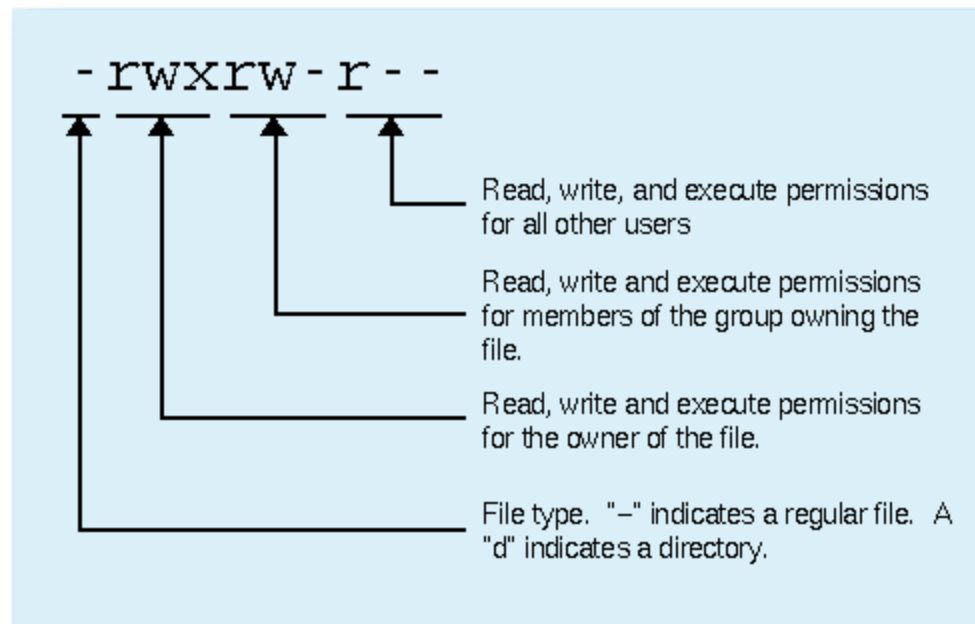
---

- Vous pouvez également attribuer les droits de la manière suivante :

☐ « `chmod u+rx test.sh` »

- u correspondant au groupe de droit « User » (utilisateur du script)
- Possibilité d'appliquer cette commande à « Group » en remplaçant « u+rx » par « g+rx », g correspondant au groupe de droit « group »

# Résumé des changements de droits possibles



# Revenons à notre script

---

- Pour l'exécuter : « `bash nom_du_script` »
- ▣ **Attention au langage de script utilisé : on utilise la commande `bash` parce que nous scriptons en `bash` !**
- Ou le rendre directement exécutable avec `chmod` :
  - ▣ « `chmod +x nom_du_script` »
  - ▣ Puis exécuter en faisant « `./nom_du_script` »

# Résultats pour test.sh

```
aymeric@ubuntu:~$ bash test.sh
Mon premier script
Liste des fichiers :
total 112
drwxr-xr-x 16 aymeric aymeric 4096 sept. 17 01:52 .
drwxr-xr-x  3 root    root    4096 sept.  8 06:34 ..
-rw-r--r--  1 aymeric aymeric   8 sept.  8 07:41 .bash_history
-rw-r--r--  1 aymeric aymeric 220 sept.  8 06:34 .bash_logout
-rw-r--r--  1 aymeric aymeric 3771 sept.  8 06:34 .bashrc
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:34 Bureau
drwxr-xr-x 13 aymeric aymeric 4096 sept.  8 09:53 .cache
drwxr-xr-x 11 aymeric aymeric 4096 sept.  8 08:34 .config
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 07:34 Documents
-rw-r--r--  1 aymeric aymeric 8980 sept.  8 06:34 examples.desktop
drwxr-xr-x  3 aymeric aymeric 4096 sept.  8 08:20 .gnupg
-rw-r--r--  1 aymeric aymeric 1272 sept. 17 01:47 .ICEAuthority
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:34 Images
drwxr-xr-x  3 aymeric aymeric 4096 sept.  8 07:34 .local
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:34 Modèles
drwxr-xr-x  5 aymeric aymeric 4096 sept.  8 07:38 .mozilla
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:34 Musique
-rw-r--r--  1 aymeric aymeric  357 sept.  8 08:26 .pam_environment
-rw-r--r--  1 aymeric aymeric  807 sept.  8 06:34 .profile
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 07:34 Public
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:20 .ssh
-rw-r--r--  1 aymeric aymeric   0 sept.  8 08:20 .sudo_as_admin_successful
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:34 Téléchargements
-rw-r--r--  1 root    root    14 sept.  8 10:57 test
-rw-r--r--  1 aymeric aymeric 240 sept. 17 01:52 test.sh
drwxr-xr-x  2 aymeric aymeric 4096 sept.  8 08:34 Vidéos
-rw-rw-r--  1 aymeric aymeric 131 sept.  8 07:42 .xinputrc
```

On constatera que la commande « `ls -la` » incluse dans le script permet de savoir quels droits sont appliqués sur chaque fichiers/dossiers du répertoire courant.



## Les variables

# Les variables

---

- En informatique, les **variables** sont des symboles qui associent un nom (l'identifiant) à une valeur. La valeur peut être de quelque type de donnée que ce soit. Le nom doit être un identifiant unique (et si le langage en possède, différents des mots-réservés ).

Source : [Wikipedia.fr/variable\\_\(Informatique\)](https://fr.wikipedia.org/wiki/Variable_(informatique))


# Les variables

---

- On peut affecter aux variables toute donnée :
  - ▢ Chaîne de caractères
  - ▢ Nombre
  - ▢ Commande
  - ▢ ...

# Variables : exemples

---

- Dans mon script test.sh, j'inclus une variable « ma\_variable » à laquelle j'associe le contenu « un\_mot »
- Cela donne :  
 ma\_variable=un\_mot
- Attention pas d'espace entre le signe « = » lors de la déclaration de la variable



# Variables : exemples

---

- Pour faire appel à une variable dans une commande, il faut inclure le signe « \$ » avant l'appel à variable :
- Cela donne la commande « echo \$ma\_variable » qui donnera la réponse « un\_mot »

Scripts Linux

---

# Les structures

# La structure « If » (Si)

- Structure permettant d'énoncer une condition nécessaire à l'exécution d'une partie du script

- Exemple :

```
if [ -f ceci_est_un_fichier ]  
then  
    echo "il s'agit bien d'un fichier"  
fi
```

- On remarque :
  - ☐ Après la condition « if », commande entre crochets.
  - ☐ Apparition de la conséquence « then » (alors)
  - ☐ Fin de structure avec « fi »

# La structure « If, Else » (Si, Sinon)

- Même principe que la structure « If » mais avec l'ajout d'une nouvelle condition « else » (sinon) :

```
else.sh
1. #!/bin/bash
2. # else example
3.
4. if [ $# -eq 1 ]
5. then
6.     nl $1
7. else
8.     nl /dev/stdin
9. fi
```

# La structure « If, Elif, Else » (Si, Sinon si, Sinon)

- Même lignée que les deux premières structures, avec l'ajout d'une 3<sup>ème</sup> condition (« elif », sinon si) qui ne servira que quand les 2 autres ne seraient pas remplies :

```
if_elif.sh
1.  #!/bin/bash
2.  # elif statements
3.
4.  if [ $1 -ge 18 ]
5.  then
6.      echo You may go to the party.
7.  elif [ $2 == 'yes' ]
8.  then
9.      echo You may go to the party but be back before midnight.
10. else
11.     echo You may not go to the party.
12. fi
```

# A noter

- A noter que les conditions Elif peuvent être empilées à l'infini tant qu'on a une condition finale :

```
#!/bin/bash
#favoritefood
if [ ${LANG:0:2} = "fr" ]; then
    echo "Vous aimez les moules frites !"
elif [ ${LANG:0:2} = "en" ]; then
    echo "You love the... pudding !"
elif [ ${LANG:0:2} = "es" ]; then
    echo "Te gusta el ramón !"
else
    echo ":'-("
fi
```

# La structure While (Tant que)



- Permet d'effectuer une boucle « tant que » une condition n'a pas été remplie
- S'écrit de la manière suivante :

```
while [ test ]  
do  
    echo 'Action en boucle'  
done
```

# Les comparateurs d'entiers



- Plusieurs paramètres peuvent être utilisés pour comparer deux entiers :
  - ▣ -eq pour une égalité
  - ▣ -ne pour une non-égalité
  - ▣ -gt pour « plus grand que » (strict)
  - ▣ -ge pour « plus grand ou égal à » (non strict)
  - ▣ -lt pour « plus petit que » (strict)
  - ▣ -le pour « plus petit ou égal à » (non strict)



# Utilisation du comparateur



```
• if [ $nombre1 -eq $nombre2 ]  
  then  
      .....  
  else  
      .....  
  fi
```