Le terminal

Manipulations basiques

La majorité des interactions avec les systèmes Linux se font à travers des terminaux. Ce premier exercice a pour objectif de vous familiariser avec le terminal, et les capacités de celui-ci.

1. Ouvrir un terminal
2. Faire afficher « bonjour ! » au terminal

```
$ > echo 'bonjour !'
```

3. Lister les fichiers du répertoire courant

```
$ > ls
```

4. Créer un répertoire « *test_dir* »

```
$ > mkdir test_dir
```

5. En une commande unique, créer un fichier nommé « *test_file* » dans le répertoire précédemment créé

```
$ > touch test_dir/test_file
```

6. Lister les fichiers contenus dans le répertoire « *test_dir* »

```
$ > ls test_dir/
```

Le fichier « test_file » devrait être visible

7. Se positionner dans le répertoire « *test_dir* »

```
$ > cd test_dir/
```

8. Modifier le fichier « *test_file* » afin d'y ajouter du texte à votre discrétion

```
$ > nano test_file
```

La commande « *nano* » permet l'édition d'un fichier.
D'autres commandes existent, comme « *vi* » par exemple. Ce sont des éditeurs de texte. Une fois la commande « *nano* » exécutée, le terminal affiche une fenêtre vide : c'est le contenu du fichier. Vous pouvez y écrire votre texte normalement. Puis, pour sauvegarder et quitter le fichier : Ctrl+X pour quitter, « Y » pour sauvegarder puis la touche « Entrée » pour valider le nom du fichier.

9. Afficher dans le terminal le contenu du fichier « *test_file* »

```
$ > cat test_file
```

Le contenu du fichier que vous avez tapé doit apparaître. Si ce n'est pas le cas, soit la sauvegarde du fichier n'a pas été faite correctement, soit vous n'avez pas précisé le bon nom de fichier dans votre commande.

10. Renommer « *test_file* » en « *test_file_v2* »

```
$ > mv test_file test_file_v2
```

Vous pouvez vérifier la bonne exécution de la commande en faisant « ls »

11. Copier « *test_file_v2* » dans un fichier « *test_file_v2_2* »

```
$ > cp test_file_v2 test_file_v2_2
```

Vous pouvez vérifier la bonne exécution de la commande en faisant « ls »

12. Supprimer « *test_file_v2* »

```
$ > rm test_file_v2
```

Vous pouvez vérifier la bonne exécution de la commande en faisant « ls »

13. Revenir dans le répertoire parent

```
$ > cd ..
```

« .. » correspond au répertoire parent.
Quel que soit le répertoire, « ls -a » affichera toujours « . » et « .. ». « . » correspond au répertoire actuel. « cd ././. » ne changera pas le répertoire où l'on se situe.

14. Supprimer le répertoire « *test_dir* »

```
$ > rm -r test_dir
```

Scripts bash

Linux permet la création de scripts bash (fichiers en *.sh*). Ceux-ci permettent l'automatisation de différentes tâches et activités. Amusons-nous par exemple à automatiser l'exercice précédent.

L'objectif ici est de pouvoir, avec une commande simple, rejouer le premier exercice de ce TD.

1. Ouvrir un terminal
2. Créer un répertoire bac à sable « *bac_a_sable* »

```
$ > mkdir bac_a_sable
```

3. Se positionner dans le répertoire « *bac_a_sable* »

```
$ > cd bac_a_sable/
```

4. Créer un fichier « *test_script.sh* »

```
$ > touch test_script.sh
```

5. Dans le fichier « *test_script.sh* » ajouter la ligne :

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

```
$ > nano test_script.sh
```

Celle-ci donne un contexte au script, elle permet de définir ce qui va exécuter le code. Ici le programme « *bash* » donc

6. Sous cette ligne, ajouter :
    - Une ligne de commande permettant la création d'un répertoire « *test_dir* »
    - Une ligne de commande permettant, en une seule commande, la création et l'écriture de « toto » dans un fichier « *test_file.txt* »
    - Une ligne de commande permettant d'afficher dans le terminal le contenu de « *test_file.txt* »
    - Une ligne supprimant le répertoire « *test_dir* » et son contenu
    - Faire afficher « *0* » au terminal

Nota : Il peut s'avérer utile de tester vos commandes avant de les ajouter à votre script…

7. Exécuter le script à l'aide de la commande :

```
 ./ test_script.sh
```

8. L'exécution du script se solde par un échec :

```
./test_script.sh
-bash: ./test_script.sh: Permission denied
```

Par défaut, un script bash est créé sans les droits d'exécution. Modifier les droits liés à ce script afin de le rendre exécutable par n'importe qui

```
$ > chmod +x test_script.sh
```

ou

```
$ > chmod a+x test_script.sh
```

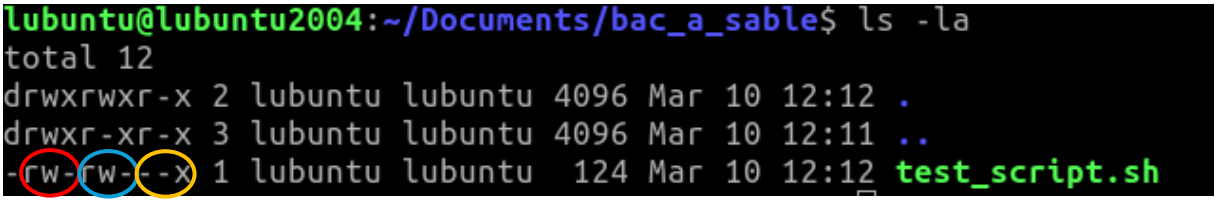
ou

```
$ > chmod 777 test_script.sh
```

Il existe plusieurs manières de changer les autorisations d'un fichier. Les autorisations sur un fichier fonctionnent par utilisateur ou groupe d'utilisateurs. Il y a trois catégories de droits sur un fichier/dossier : les droits du propriétaire (souvent celui qui a créé le fichier), les droits du groupe d'utilisateurs auquel le fichier appartient et les autres. En

anglais, on parle de « user », « group » and « other ». On peut donc attribuer des droits selon ces catégories.

Il est aussi possible de donner des droits aux utilisateurs directement en octet, c'est le cas de la dernière ligne : le premier octet est pour le « user », le deuxième pour le « group » et le troisième pour « other ». Ensuite, pour savoir quelle valeur mettre, il faut savoir que cet octet est séparé en 3 bits (2^3 = 8 bits = 1 octet) : le bit de poids fort est pour le droit d'écriture « read (r) », le bit du milieu pour le droit d'écriture « write (w)» et le bit de poids faible pour le droit d'exécution « execute (x) ». On retrouve ces informations en faisant ls -la :



En rouge : les droits du owner, lire et écrire. En bleu les droits du groupe, lire et écrite. Et en jaune les droits des autres, exécuter.

9. Lancer le script « *test_script.sh* » :

```
$ ./test_script.sh
toto
0
```

```
# !/bin/bash

mkdir test_dir
echo 'toto' > test_dir/test_file.txt
cat test_dir/test_file.txt
rm -r test_dir
echo '0'
```

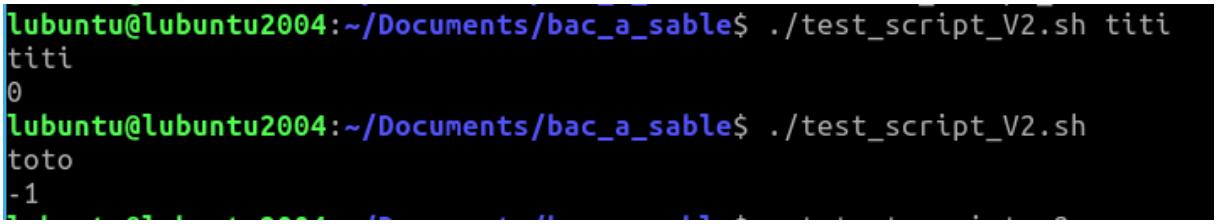
10. Pousser plus loin les capacités du script :
    - Permettez de donner du texte en argument au script. Celui-ci écrira ce texte dans le fichier « *test_file.txt* ».
    - Ajouter une condition au script : si on n'a pas donné de texte en argument du script, le texte est « toto » par défaut et le script retourne « -1 » au lieu de « 0 »

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

```
if [ $# -eq 0 ]
then
      text_to_write=toto
      value_to_return=-1
else
      text_to_write=$1
      value_to_return=0
fi

mkdir test_dir
echo $text_to_write > test_dir/test_file.txt
cat test_dir/test_file.txt
rm -r test_dir
echo $value_to_return
```

Résultat :



Exercice 1 :

```
#!/bin/bash

fichier="FichierNote.txt"

while read -r ligne; do
    stringarray=($ligne)
    note=${stringarray[2]}
    if [ $note -ge "10" ]; then
        echo "$ligne"
    fi
done < $fichier
```

Exercice 2 :

Script1.sh

```
#!/bin/bash

# Générer 10 nombres aléatoires et les écrire dans nombres.txt
for ((i=1; i<=10; i++))
do
     echo $((RANDOM % 100 + 1)) >> nombres.txt
done
```

Script2.sh

```
#!/bin/bash

# Lire les nombres à partir de nombres.txt, calculer la somme et
écrire dans somme.txt

sum=0
while read number
do
     sum=$((sum + number))
done < nombres.txt
echo "La somme des nombres est : $sum" > somme.txt
```

Annexe A : Commandes et redirection

Les commandes Linux acceptent de nombreuses options dont on peut consulter la documentation en tapant `man ls` par exemple pour la commande `ls`. Celle-ci comporte des options pour affiche les fichiers cachés `ls -a` ou encore pour afficher les détails et permissions d'un fichier `ls -l`.

Commande	Description
`cd`	*Change Directory* Se déplacer dans l'arborescence
`ls`	*Lister* le contenu du répertoire courant
`mkdir`	Permet de créer un répertoire ***(make dir)***
`pwd`	*Affiche le dossier courant* (Print Workgin Directory)
`cp`	*Copier* des fichiers ou des répertoires
`mv`	*Déplacer (move)* ou renommer des fichiers ou des répertoires
`rm`	*Effacer (remove)* des fichiers ou des répertoires
`cat`	*Visualiser (concaténer)* le contenu d'un fichier
`echo`	*Afficher* un message ou le contenu d'une variable
`touch`	*Créer* un fichier vide ou réinitialiser le *timestamp* d'un fichier
`chmod`	Modifier les autorisations d'accès à un fichier
`nano`	*Éditeur de texte* en ligne de commande. Il y en a beaucoup
`ps`	Afficher les informations des *processus* en cours
`top`	*Gestionnaire de ressource* en TUI
`kill`	Envoyer un *signal* au processus, généralement pour l'arrêter
`grep`	*Filtrer* une sortie en ne gardant que les lignes contenant un terme

Linux permet la redirection des sorties des commandes.

Une commande `echo 'bonjour'` aura pour sortie « *bonjour* ». Cette sortie peut être redirigée afin de ne pas être affichée du tout, ou au contraire pour être dirigée vers un fichier texte, ou de log par exemple :

```
$ echo 'bonjour'
bonjour
$ echo 'bonjour' > /dev/null
$ echo 'bonjour' > text.txt
$ cat text.txt
Bonjour
```

Nota : rediriger une sortie vers un fichier inexistant aura pour effet de créer ce fichier.