



Table des matières

I Le système Linux	2
I.1 Histoire	2
I.2 Développement	2
I.3 Installation	2
II Le Bash	2
II.1 Les commandes de bases Linux ou Windows PowerShell	3
II.2 Naviguer dans une arborescence	4
II.3 Quelques exemples	5
III Ligne de commande et motif glob	6
IV Entrées/Sorties en shell Bash	8
V Les filtres, tubes (pipes) et redirections	8
VI Droits et permissions sous Unix	9
VI.1 Droits et groupes	9
VI.2 Changer les droits	10
VII script Bash (Hors programme mais pour les vrais curieux)	12
VII.1 Les quotes	12
VII.2 Enregistrer une variable	13
VII.3 Ecrire un script	13
VII.4 Les paramètres	14
VII.5 Les tableaux	14
VII.6 Les tests if	15
VII.7 Opérateurs logiques	15
VII.8 Les boucles	16
VII.9 Pour continuer à s'entraîner	16

Extraits du programme

- **B.O. : "Modèle d'architecture séquentielle (von Neumann) :**
Distinguer les rôles et les caractéristiques des différents constituants d'une machine. Dérouler l'exécution d'une séquence d'instructions simples du type langage machine. La présentation se limite aux concepts généraux. On distingue les architectures monoprocesseur et les architectures multiprocesseur. Des activités débranchées sont proposées. Les circuits combinatoires réalisent des fonctions booléennes."
- **B.O. : "Systèmes d'exploitation :**
Identifier les fonctions d'un système d'exploitation. Utiliser les commandes de base en ligne de commande. Gérer les droits et permissions d'accès aux fichiers. Les différences entre systèmes d'exploitation libres et propriétaires sont évoquées. Les élèves utilisent un système d'exploitation libre. "

Objectifs

- Une introduction au système Linux.
- Initiation à l'usage du Bash, l'interpréteur de commandes de ce système.
- Identifier les moyens de naviguer dans l'arborescence d'un système.
- Manipuler les entrées/sorties et les redirections.
- Gérer les droits et permissions d'accès aux fichiers.

I Le système Linux

I.1 Histoire

- Le 27 septembre 1983, Richard Stallman dévoile son projet de développer un système d'exploitation compatible UNIX appelé **GNU** - acronyme récursif (la forme développée du sigle contient sa forme réduite) qui signifie en anglais " GNU's Not UNIX " (littéralement, " GNU n'est pas UNIX ") , en invitant la communauté hacker à le rejoindre et participer à son développement.
- Le noyau **Linux** a été créé en 1991 par **Linus Torvalds**, un informaticien américano-finlandais né le 28 décembre 1969 à Helsinki en Finlande.
- Le manchot Tux, dessiné par Larry Ewing en 1996, devient la mascotte du projet.
- À l'origine, le noyau **Linux** a été développé pour les ordinateurs personnels compatibles PC, et devait être accompagné des logiciels GNU pour constituer un système d'exploitation.
- Depuis les années 2000, le noyau Linux est utilisé sur du matériel informatique allant des téléphones portables aux super-ordinateurs, et n'est pas toujours accompagné de logiciels GNU. C'est notamment le cas d'Android, qui équipe plus de 80% des smartphones.

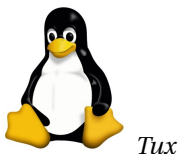


Linus Torvalds

I.2 Développement

Linux s'est développé rapidement et a donné naissance à de nombreux systèmes d'exploitation partageant le même noyau dont Ubuntu, Debian et le système Android.

Le noyau Linux est écrit en langage C et contient des millions de lignes de code.



Tux



I.3 Installation

L'installation de Linux a longtemps été réservée à des experts mais cela est bien plus aisé de nos jours.

De nombreuses personnes choisissent de configurer leurs machines et sous Windows, et sous Linux, l'utilisateur choisi lors du démarrage sur quel système il veut travailler.

II Le Bash

Nous allons ici présenter le Bash (Bourne Again Shell), l'interpréteur de commandes le plus courant sous Linux, sous macOS et même sous Windows 10 via le **PowerShell** (ou la commande cmd dans l'invite de commandes).

II.1 Les commandes de bases Linux ou Windows PowerShell

Commandes Unix	PowerShell (si différent)	Description	Exemple
cd		Se déplacer dans l'arborescence (ou parfois chdir, abréviation de <i>change directory</i>)	cd home\NSI\eleve
cd ..		Se déplacer dans l'arborescence dans le répertoire parent	cd ..
pwd		Cette commande permet d'afficher l'emplacement où on se situe actuellement dans la hiérarchie FHS (<i>Filesystem Hierarchy Standard</i>).	pwd
rmdir		Cette commande permet de supprimer un répertoire.	rmdir NSI
mkdir		Cette commande permet de créer un répertoire.	mkdir NSI
cd ~ (ou cd)	cd\	Se déplacer dans le répertoire d'accueil (HOME sous Linux) des utilisateurs. Sous Windows c'est souvent un prénom ou un dossier Public	cd ~ (ou cd)
cat	type	Visualiser le contenu d'un fichier (<i>catenate</i> , synonyme de <i>concatenate</i> : « concaténer »)	cat fichier1
cp	cpi	Copier des fichiers ou des répertoires (de <i>copy item</i>) <i>Remarque : on peut donner le chemin absolu du fichier copie si il n'est pas dans le même répertoire</i>	cp fichier1 copie2 cpi fichier1 copie2
echo		Afficher un message ou le contenu d'une variable (remplace le printf, print formatted)	echo "Bonjour"
ls	dir	Lister le contenu du répertoire courant (de <i>list</i> en anglais)	ls
mv	mi	Déplacer des fichiers ou des répertoires (de <i>Move-Item</i>)	mv fichier1 deplace2
mv	rni	Renommer fichiers ou répertoires (de <i>ReName-Item</i>)	mv fichier1 rename2 rni fichier1 rename2
rm	ri	Effacer des fichiers ou des répertoires	
touch	echo \$null >>	Actualise la date d'accès et/ou de modification d'un fichier (le <i>timestamp</i>). Si le fichier n'existe pas, est créé un fichier vide. Sous PowerShell on utilise echo \$null >> qui ajoute une ligne vide au fichier.	touch fichier echo \$null >> fichier
man	help	Permet d'avoir la description d'une commande	man cd
tracert	tracert	Pour connaître l'itinéraire vers une destination sur un réseau	tracert www.math93.com



Remarque

De nombreux compléments et exemples sur les sites :

- <https://www.sitedetout.com/tutoriels/commandes-linux-de-base/>
- https://windows.developpez.com/cours/ligne-commande/?page=page_4
- https://en.wikipedia.org/wiki/List_of_Unix_commands
- **Compatibilité Unix / PowerShell** : <https://urlz.fr/bugu>

II.2 Naviguer dans une arborescence



Naviguer dans une arborescence

Soit l'arborescence suivante (ce sont des répertoires) : **C:\Users\lycee\nsi\eleve**

- Pour aller dans le répertoire **eleve** situé dans le répertoire **nsi** à partir du répertoire **lycee** on utilise soit :
 - un "chemin relatif" qui part du répertoire où l'on se trouve : **cd NSI\eleve**
 - ou un chemin absolu c'est à dire un chemin qui part de la racine on peut faire : **cd C:\Users\lycee\nsi\eleve**
- Pour remonter dans le répertoire parent on utilise deux points ".." : **cd ..**
- Si on veut copier le fichier a.txt du répertoire nsi, dans le répertoire eleve alors que l'on se trouve dans le répertoire eleve on peut faire : **cp . / a.txt b.txt**

	cp	./ a.txt	b.txt
instruction	chemin fichier 1		chemin fichier 2

II.3 Quelques exemples

— Exemple 1.

1. On affiche "Hello world!" : **echo "Hello world!"**
2. On se déplace dans le dossier *lycee* : **cd lycee**
3. On crée le dossier *NSI* : **mkdir NSI**

```

1 PS C:\Users> echo "Hello world !"
2 Hello world !
3 PS C:\Users> cd lycee
4 PS C:\Users\lycee> mkdir NSI
5
6
7     Directory: C:\Users\lycee
8
9
10  Mode                LastWriteTime         Length Name
11  ----                -
12  d-----            04/01/2020    18:24         NSI
13

```

— Exemple 2.

1. On se déplace dans le dossier *NSI* : **cd NSI**
2. On écrit "Bonjour" dans le fichier *test.txt* : **echo "Bonjour" > test.txt**

```

1 PS C:\Users\lycee> cd NSI
2 PS C:\Users\lycee\NSI> echo "Bonjour" > test.txt

```

— Exemple 3.

1. On affiche le contenu du dossier courant *NSI* : **ls**
2. On affiche le contenu du fichier *test.txt* : **cat test.txt**

```

PS C:\Users\lycee\NSI> ls

    Directory: C:\Users\franc\NSI

Mode                LastWriteTime         Length Name
----                -
-a-----            04/01/2020    18:26         20 test.txt




PS C:\Users\lycee\NSI> cat test.txt
Bonjour

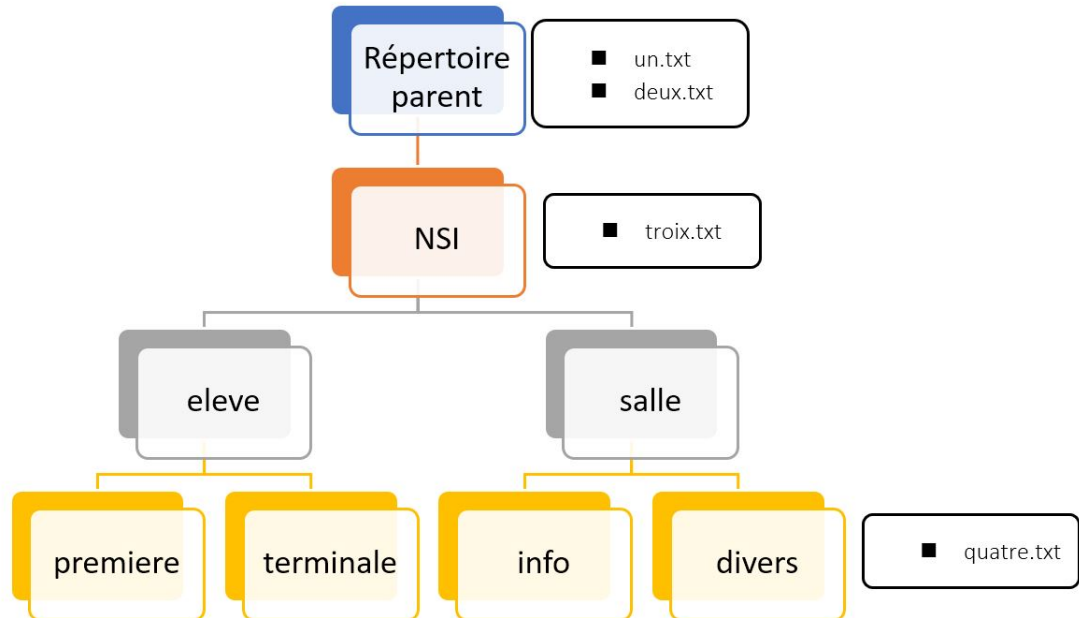
```



Exercice 1

Remarque : vous pouvez réaliser cet exercice sur

- repl.it en créant un fichier bash  (au lieu de python).
 - [Capytale](#) en ouvrant une console bash en cliquant sur  puis .
1. Créer dans le répertoire d'accueil (répertoire parent dans le schéma), l'arborescence ci-dessous constituée des 7 répertoires suivants : NSI, eleve, salle, premiere, terminale, info, divers.



2. Créer deux fichiers vides appelés « *un.txt* » et « *deux.txt* » dans votre répertoire d'accueil (le parent de NSI). Utilisez la commande : `echo "" > fichier.txt`
3. Écrire le texte "La NSI, c'est ma passion!" dans le fichier « *un.txt* ».
4. Vous déplacer dans votre répertoire **eleve**.
Afficher le contenu du fichier « *un.txt* », à partir de ce répertoire en utilisant un chemin absolu du fichier.
5. Déplacer vous dans le répertoire **NSI**.
En utilisant un chemin relatif, copier le fichier « *un.txt* » dans le répertoire **NSI** en lui donnant le nom « *trois.txt* ».
6. Déplacer vous dans le répertoire **salle**.
En utilisant un chemin absolu, copier le fichier « *trois.txt* » dans le répertoire **divers** en lui donnant le nom « *quatre.txt* ».

III Ligne de commande et motif glob

1. Motifs génériques.

Une chaîne est un motif générique si elle contient un ou plusieurs caractères parmi « ? », « * » et « [». Le développement (**globbing**) est l'opération qui transforme un motif générique en une liste de noms de fichiers correspondant à ce motif. La correspondance est définie ainsi :

- Un « ? » (sans les guillemets) correspond à n'importe quel caractère.
- Un « * » (sans les guillemets) correspond à n'importe quelle chaîne, y compris la chaîne vide.

2. Exemple 1

Si on souhaite par exemple afficher tous les noms de fichiers dont l'extension est .jpg on peut utiliser :

```
ls *.jpg
```

3. Exemple 2

Si on souhaite par exemple afficher tous les noms de fichiers qui commencent par `img` et se terminent par `.txt` on peut utiliser :

```
ls img*.txt
```

4. Exemple 3

Si on souhaite par exemple afficher tous les noms de fichiers qui sont de la forme `nom?.txt` où `?` est un seul caractère, on peut utiliser :

```
ls nom?.txt
```



Exercice 2

On vous donne une série de commande sous une console bash. Le `$` indique que vous êtes dans un répertoire racine mais cela n'est pas important pour l'exercice.

```
$> ls
entier.py
flottant.py
readme.md
$> mkdir foo
$> mv *.py foo
```

Question 1

Que peut-on dire du système de fichiers suite à l'exécution ci-dessus ?

- a. Les fichiers **entier.py**, **flottant.py** et **foo** ont été déplacés dans le répertoire de l'utilisateur
- b. L'utilisateur **foo** est propriétaire des fichiers **entier.py** et **flottant.py**
- c. Le répertoire **foo** contient le résultat de l'exécution des deux fichiers **entier.py** et **flottant.py**
- d. Le répertoire **foo** contient deux fichiers d'extension **.py**

IV Entrées/Sorties en shell Bash

Objectifs

- Manipuler les entrées/sorties et les redirections

1. Les entrées : read et Read-Host

Pour effectuer une saisie utilisateur et afficher le contenu de la variable.

```
# Sous Unix
read Age
echo $Age

# Sous PowerShell
$Age = Read-Host "Quel est votre âge s'il vous plaît"
echo $Age
```

2. Les entrées sorties dans un fichier : > et >>

- **echo "Bonjour" > salut.txt :**

Pour envoyer le mot "Bonjour" dans le fichier *salut.txt* qui est créé (ou réinitialiser, avec contenu effacé) .

- **echo "Comment ça va?" >> salut.txt :**

Pour ajouter une nouvelle ligne "Comment ça va?" dans le fichier existant *salut.txt*.

```
echo "Bonjour" > salut.txt
echo "Comment ça va ?" >> salut.txt
```

V Les filtres, tubes (pipes) et redirections

Unix permet l'utilisation de filtres (comme *wc*, *sort*, *cut*, *tr*) qui sont souvent combinés avec des tubes (pipes) notés "|" pour envoyer le résultat dans un fichier. Les lettres précédées d'un tiret sont appelées des *flags* et permettent de choisir des options.

1. Compter des lignes ou des caractères : wc ou gc (Get-Content)

```
# Sous Unix
wc -l test.txt # pour compter les lignes
wc -c test.txt # pour compter les caractères

# Sous PowerShell
# pour compter les lignes (gc ou Get-Content)
Get-Content test4.txt | Measure-Object -Line

# pour compter les caractères (gc ou Get-Content)
gc test4.txt | Measure-Object -Character
```

2. Pour trier : sort

Pour trier ligne par ligne le fichier *test4.txt* et envoyer le fichier trié dans *test5.txt*.

```
cat test4.txt | sort > test5.txt
cat test5.txt
```




Exercice 3

Copier-coller le poème de Baudelaire ci-dessous dans un fichier **poeme.txt**. Avec des commandes bash :

1. Compter le nombre de lignes;
2. Compter le nombre de caractere;
3. Trier les lignes et mettre le résultat dans un fichier **trier.txt**. Regarder le résultat.

Souvent, pour s'amuser, les hommes d'équipage
 Prennent des albatros, vastes oiseaux des mers,
 Qui suivent, indolents compagnons de voyage,
 Le navire glissant sur les gouffres amers.

A peine les ont-ils déposés sur les planches,
 Que ces rois de l'azur, maladroits et honteux,
 Laissent piteusement leurs grandes ailes blanches
 Comme des avirons traîner à côté d'eux.

Ce voyageur aillé, comme il est gauche et veule !
 Lui, naguère si beau, qu'il est comique et laid !
 L'un agace son bec avec un brûle-gueule,
 L'autre mime, en boitant, l'infirme qui volait !

Le Poète est semblable au prince des nuées
 Qui hante la tempête et se rit de l'archer ;
 Exilé sur le sol au milieu des huées,
 Ses ailes de géant l'empêchent de marcher.

VI Droits et permissions sous Unix

VI.1 Droits et groupes

1. Les groupes

Unix sépare le monde en 3 groupes et chaque fichier (ou répertoire) va préciser les droits attribués à chacun de ces groupes.

- l'utilisateur ou propriétaire (user) ;
- le groupe (group) ;
- le reste (others ou public) .

2. Les droits

Les droits sur un fichier UNIX s'attribuent sur trois « actions » différentes possibles :

- la lecture (r) : on peut par exemple lire le fichier avec un logiciel.
 Lorsque ce droit est alloué à un répertoire, il autorise l'affichage du contenu du répertoire (la liste des fichiers présents à la racine de ce répertoire).
- l'écriture (w) : on peut modifier le fichier et le vider de son contenu.
 Lorsque ce droit est alloué à un répertoire, il autorise la création, la suppression et le changement de nom des fichiers qu'il contient, quels que soient les droits d'accès des fichiers de ce répertoire (même s'ils ne possèdent pas eux-mêmes le droit en écriture).
- l'exécution (x) : on peut exécuter le fichier s'il est prévu pour, c'est-à-dire si c'est un fichier exécutable.
 Lorsque ce droit est attribué à un répertoire, il autorise l'accès (ou ouverture) au répertoire.

On appelle parfois **r**, **w** et **x** des « **flags** » ou « **drapeaux** ». Sur un fichier donné, ces 3 flags doivent être définis pour son propriétaire, son groupe, mais aussi les autres utilisateurs (différents du propriétaire et n'appartenant pas au groupe).

3. Représentation des droits.

- Cet ensemble de 3 droits sur 3 entités se représente généralement de la façon suivante : on écrit côte à côte les droits r, w puis x respectivement pour le propriétaire (u), le groupe (g) et les autres utilisateurs (o).
- Les codes u, g et o (u comme user, g comme group et o comme others) sont utilisés par les commandes UNIX qui permettent d'attribuer les droits et l'appartenance des fichiers. Lorsqu'un flag est attribué à une entité, on écrit ce flag (r, w ou x), et lorsqu'il n'est pas attribué, on écrit un « - ».

4. En ligne de commande

- Les droits des fichiers d'un répertoire peuvent être affichés par la commande : **ls -l**
Attention il n'y a que des L minuscules et pas le chiffre un.

```
ls -l test.txt
```

- Les droits d'accès apparaissent alors comme une liste de 10 symboles :

```
drwxr-xr-x
```

- Le premier symbole peut être « - », « **d** », soit « **l** », entres autres . Il indique la nature du fichier :
 - fichier classique : -
 - répertoire : **d**
 - lien symbolique : **l**
- Dans cet exemple :

```
drwxr-xr-x
```

```
drwxr-xr-x => [d] [rwx] [r-x] [r-x]
```

- **[d]** : c'est un répertoire.
- **[rwx]** pour le 1er groupe de 3 symboles : son propriétaire peut lire, écrire et exécuter.
- **[r-x]** pour le 2nd groupe de 3 symboles : le groupe peut uniquement lire et exécuter le fichier, sans pouvoir le modifier.
- **[r-x]** pour le 3ème groupe de 3 symboles : le reste du monde peut uniquement lire et exécuter le fichier, sans pouvoir le modifier.
- En pratique, en exécutant la commande suivante : **ls -l**
on obtient la liste du contenu du répertoire courant, par exemple :

```
drwxr-xr-x 6 roza lycee 4096 2019-10-29 23:09 Bureau
drwxr-x--- 2 roza lycee 4096 2019-10-22 22:46 Documents
lrwxrwxrwx 1 roza lycee 26 2019-09-22 22:30 Examples -> /usr/share/ex
-rw-r--r-- 1 roza lycee 1544881 2019-10-18 15:37 forum.xcf
drwxr-xr-x 7 roza lycee 4096 2019-09-23 18:16 Images
```

On retrouve dans la première colonne le groupe de 10 caractères permettant de connaître les droits pour chaque fichier.

Ainsi, pour le fichier forum.xcf, on a : **-rw-r--r--** soit **[-] [rw-] [r - -] [r - -]**

- Le 1er caractère est - : c'est un fichier.
- Le premier groupe de 3 caractères est **[rw-]** :
Le propriétaire roza a le droit de lecture et écriture (mais pas d'exécution) sur le fichier.
- Les 2 groupes suivants sont **[r - -]** :
Les utilisateurs du groupe lycee et les autres n'ont que le droit de lecture (pas d'écriture, ni d'exécution) .
- Bilan : Ce fichier appartient à l'utilisateur roza qui a les droits d'écritures et lecture, et est accessible en lecture au groupe lycee et aux autres

VI.2 Changer les droits

Seul le propriétaire d'un fichier peut changer les permissions d'accès.

- L'outil **chmod** (change mode, changer les permissions) permet de modifier les permissions sur un fichier. Il peut s'employer de deux façons :
 - soit en précisant les permissions de manière octale, à l'aide de chiffres);
 - soit en ajoutant ou en retirant des permissions à une ou plusieurs catégories d'utilisateurs à l'aide des symboles r w et x, que nous avons présentés plus haut.

— Avec la deuxième méthode, on va choisir :

. À qui s'applique le changement :

- u (user, utilisateur) représente la catégorie "propriétaire";
- g (group, groupe) représente la catégorie "groupe propriétaire";
- o (others, autres) représente la catégorie "reste du monde";
- a (all, tous) représente l'ensemble des trois catégories.

. La modification que l'on veut faire

- + : ajouter
- - : supprimer
- = : affectation

. Le droit que l'on veut modifier

- r : read : lecture
- w : write : écriture
- x : execute : exécution, autorisation d'exécution. La permission d'exécution régit également l'accès à un répertoire : si l'exécution n'est pas autorisée sur un répertoire, on ne peut faire un `chdir` (commande `cd`) sur ce répertoire.

— Par exemples, on enlèvera le droit d'écriture pour les autres avec :

```
chmod o-w fichier3
```

— Par exemples, ajoutera le droit d'exécution à tout le monde avec :

```
chmod a+x fichier3
```

— On peut aussi combiner plusieurs actions en même temps :

- . On ajoute la permission de lecture, d'écriture et d'exécution sur le fichier `fichier3` pour le propriétaire;
- . On ajoute la permission de lecture et d'exécution au groupe propriétaire, on retire la permission d'écriture;
- . On ajoute la permission de lecture aux autres, on retire la permission d'écriture et d'exécution.

```
chmod u+rw, g+rx-w, o+r-wx fichier3
```

— **En Octal.**

En octal, chaque « groupement » de droits (pour user, group et other) sera représenté par un chiffre et à chaque droit correspond une valeur :

- . r (read) = 4
- . w (write) = 2
- . x (execute) = 1
- . - = 0

Par exemple,

- . Pour `rw`, on aura : $4+2=6$
- . Pour `rw-`, on aura : $4+2+0=6$
- . Pour `r-`, on aura : $4+0+0=4$

Ce qui permet de faire toutes les combinaisons :

- . 0 : - - - (aucun droit)
- . 1 : - - x (exécution)
- . 2 : - w - (écriture)
- . 3 : - w x (écriture et exécution= $2+1$)
- . 4 : r - - (lecture seule)
- . 5 : r - x (lecture et exécution= $4+1$)

- . 6 : r w - (lecture et écriture=4+2)
- . 7 : r w x (lecture, écriture et exécution=4+2+1)

— Exemple : Reprenons le répertoire Documents. Ses permissions sont :

```
drwxr-x---
```

En octal, on aura 750 :

rwx	r-x	- - -
7(4+2+1)	5(4+0+1)	0(0+0+0)

Pour mettre ces permissions sur le répertoire on taperait donc la commande :

```
chmod 750 Documents
```



Exercice 4

En exécutant la commande suivante : **ls -l**

on obtient la liste du contenu du répertoire courant ci-dessous :

1. Donner le nom de l'utilisateur auquel appartient le fichier1, les droits qu'il a sur ce fichier, ceux du groupe et des autres.
2. Faire de même pour les fichiers 2 et 3.
3. Donner l'équivalent en octal du droit correspondant.
4. Pour le le fichier **fichier3**, donner la commande avec la méthode octale puis la méthode utilisant les symboles(r, w, x, u, g, a ...) qui donne
 - pour le propriétaire, le droit en écriture et supprime le droit en exécution;
 - pour le groupe, les droits en lecture et écriture;
 - pour le public (le reste) le droit en lecture.

```
-rwx----- 1 roza lycee      4096 2019-10-29 23:09 fichier1
-rwx--x--x  1 pierre admin   4096 2019-10-22 22:46 fichier2
-r-xr----- 1 alice etu      26 2019-09-22 22:30 fichier3
```

↩️ Fin du cours ➡️

VII script Bash (Hors programme mais pour les vrais curieux)

VII.1 Les quotes

- Simple quote ou apostrophe (touche du 4)

Les simples quotes délimitent une chaîne de caractères. Même si cette chaîne contient des commandes ou des variables shell, celles-ci ne seront pas interprétées. Par exemple :

```
$ variable="secret"
$ echo 'Mon mot de passe est $variable.'
Mon mot de passe est $variable.
```

- Doubles quotes ou guillemets (touche du 3)

Les doubles quotes délimitent une chaîne de caractères, mais les noms de variable sont interprétés par le shell. Par exemple :

```
$ variable="secret"
$ echo "Mon mot de passe est $variable."
Mon mot de passe est secret.
```

Ceci est utile pour générer des messages dynamiques au sein d'un script.

- Back-quote, apostrophe inversée ou accent grave (Alt Gr + 7)

Bash considère que les Back-quotes délimitent une commande à exécuter. Les noms de variable et les commandes sont donc interprétés. Par exemple :

```
$ echo `variable="connu"; echo "Mon mot de passe est $variable."`
Mon mot de passe est connu.
```

Autre exemple :

```
echo `ls`
```



Exercice 5

- | Exécuter les commandes suivantes, changer les quotes pour voir les différents résultats :

```
message='Bonjour tout le monde'
echo 'Le message est : $message'
message="Bonjour tout le monde"
echo "Le message est : $message"
message=`pwd`
echo "Vous êtes dans le dossier $message".
```

VII.2 Enregistrer une variable

La commande **read** joue le rôle de 'input'. Le flag **-p** sert à associer un message.

```
read prenom nom
echo $prenom $nom
read -p 'entrez votre nom :' nom
echo $nom
```

VII.3 Ecrire un script

Pour enregistrer un script bash, il faut ouvrir un éditeur et mettre le format **.sh** comme par exemple :

```
toto.sh
```

Pour lancer le script en ligne de commande vous devez taper :

```
./toto.sh
```

Un script de base s'écrit comme suit ; remarquer l'en-tête nécessaire **#!/bin/bash**

```
#!/bin/bash
let "a = 5"
let "b = 2"
let "c = a + b"
echo $c
```

**Exercice 6**

- Rédiger le script précédent dans **toto.sh**, lancer le, changer les quotes pour voir les différents résultats.

**Remarque**

Attention les espaces sont fondamentales dans un script bash, si votre script ne marche pas, vérifier d'abord les espaces.
Parfois le fichier ne se lance pas parce que vous n'avez pas les droits, n'hésitez pas à vous attribuer les droits d'exécution avec **'chmod'**.

VII.4 Les paramètres

On peut lancer un script en mode commande avec des paramètres comme par exemple :

```
./toto.sh 25 bonjour 47
```

```
./variables.sh param1 param2 param3
$# : contient le nombre de paramètres ;
$0 : contient le nom du script exécuté (ici ./variables.sh) ;
$1 : contient le premier paramètre ;
$2 : contient le second paramètre ;
...
```

**Exercice 7**

- Rédiger le script suivant appelé **addition.sh** et lancer le :

```
#!/bin/bash
let "c = $1 + $2"
echo "$1 + $2"$c
echo "le script s'appelle $0 et contient $# paramètres"
```

**Exercice 8**

- Écrire un script nommé **bonjour.sh** qui prend vos prénom et nom en paramètre et écrit "Bonjour [Prénom] [Nom]".

VII.5 Les tableaux

```
#!/bin/bash
tableau=('valeur0' 'valeur1' 'valeur2')
tableau[5]='valeur5'
echo ${tableau[*]}
echo " le troisieme element est" ${tableau[2]}
```

**Exercice 9**

- Reprendre le script **bonjour.sh** précédent, copier les paramètres dans un tableau, puis afficher le tableau (Bonjour « prénom » « nom »!).

VII.6 Les tests if

La structure ressemble beaucoup au python sauf le 'fi' à la fin.

```
#!/bin/bash
if [ test ]
then
    echo "Le premier test a été vérifié"
elif [ autre_test ]
then
    echo "Le second test a été vérifié"
elif [ encore_autre_test ]
then
    echo "Le troisième test a été vérifié"
else
    echo "Aucun des tests précédents n'a été vérifié"
fi
```



Exercice 10

Ecrire le script suivant sans copier-coller. Si vous ne faites aucune erreur en l'écrivant vous êtes très fort. Remarquer que l'on peut éviter les passages à la ligne avec des points-virgules.

Attention, en bash les espaces sont importants. Vous devez faire attention aux espaces après le crochet ouvert ([) et avant le crochet fermé (]). Mais aussi ne les oubliez pas non plus avant et après le signe égal (=).

```
#!/bin/bash
read -p 'qui es-tu ?' nom
if [ $nom = "Bruno" ]; then
    echo "Salut Bruno !"
else
    echo "Je ne vous connais pas!"
fi
```

VII.7 Opérateurs logiques

- && pour le ET logique
- || pour le OU logique
- ! (point d'exclamation) pour la négation.



Exercice 11

Ecrire le script suivant sans copier-coller.

```
#!/bin/bash
read -p "Si vous etes d'accord entrez o ou oui : " reponse
if [ ! $reponse = "o" ] && [ ! $reponse = "oui" ]; then
    echo "Non, je ne suis pas d'accord !"
else
    echo "Oui, je suis d'accord"
fi
```

VII.8 Les boucles

Voici quelques exemples

```
#!/bin/bash
while [ -z $reponse ] || [ $reponse != 'oui' ]
do
    read -p 'Dites oui : ' reponse
done
```

```
#!/bin/bash
for variable in 'valeur1' 'valeur2' 'valeur3'
do
    echo "La variable vaut $variable"
done
```

```
#!/bin/bash
for animal in 'chien' 'souris' 'moineau'
do
    echo "Animal en cours d'analyse : $animal"
done
```



Exercice 12

| Que va faire le script suivant?

```
#!/bin/bash

for fichier in `ls`
do
    mv $fichier $fichier-old
done
```

VII.9 Pour continuer à s'entraîner

Faire les exercices proposés au lien suivant : <https://ineumann.developpez.com/tutoriels/linux/exercices-shell/>
Ne regardez pas tout de suite les solutions.