Travaux Pratiques -- Scripts Systèmes

v1.2

Table des matières

TP 1 – Introduction aux scripts Bash – 1

Exercice 1

a) Que fait le script Bash suivant ?

```
#!/bin/bash

for i in {1..50}
do
    mkdir toto$i
done
```

b) Sauver le script sous « 1.sh ».

c) Faire « bash 1.sh ».

d) Faire « chmod +x 1.sh » puis « ./1.sh ».

Exercice 2

Que fait le script Bash suivant ?

```
#!/bin/bash

for i in {1..50}
do
    rmdir toto$i
done
```

Exercice 3

a) Que fait le script Bash suivant ?

```
#!/bin/bash

for i in {1..5}
do
    mkdir toto$i
    cd toto$i
done
```

b) Faire « ls ».

c) Faire « tree toto1/ ».

d) Effacer « toto1/ ».

Exercice 4

a) Que fait le script Bash suivant ?

```
#!/bin/bash

mkdir test_script
cd test_script
for i in {1..50}
do
    touch info_$i.txt
done
```

b) Faire « ls test_script/ ».

c) Effacer « test_script/ ».

Exercice 5

a) Que fait le script Bash suivant ?

```
#!/bin/bash

mkdir test_script
cd test_script
for i in {1..50}
do
    echo toto$i > info_$i.txt
done
```

b) Faire « ls test_script/ ».

c) Faire « cat test_script/info_1.txt ».

d) Faire « cat test_script/info_50.txt ».

e) Que fait le script Bash suivant ?

```
#!/bin/bash

cd test_script
```

```
for i in {1..50}
do
    cat info_$i.txt
done
```

f) Effacer « test_script/ ».

Exercice 6

a) Que fait le script Bash suivant ?

```
#!/bin/bash

for i in {1..50}
do
    if [ "$i" = "$1" ]
    then
        echo "trouvé"
    fi
done
```

b) Faire « ./6.sh ».

c) Faire « ./6.sh toto».

d) Faire « ./6.sh 1».

e) Faire « ./6.sh 50».

f) Faire « ./6.sh 100».

Exercice 7

Que fait le script Bash suivant ?

```
#!/bin/bash

res=0

for i in {1..50}
do
    if [ "$i" = "$1" ]
    then
        echo "trouvé"
```

```
        res=1
    fi
done

if [ "$res" = 0 ]
then
    echo "non trouvé"
fi
```

TP 2 – Introduction aux scripts Bash – 2

Exercice 1

Que fait le script Bash suivant ?

```
#!/bin/bash

mkdir test_script
cd test_script

for i in {1..50}
do
    mkdir toto$i
    echo toto$i > toto$i/info.txt
done

mkdir info

for i in {1..50}
do
    mv toto$i/info.txt toto$i/info_$i.txt
done

for i in {1..50}
do
    mv toto$i/info_$i.txt info
done
```

Exercice 2

Corrigez le script Bash suivant ?

```
#!/bin/bash

mkdir test_script
cd test_script

for i in {1..50}
do
    mkdir toto$i
    echo toto$i > toto$i/info.txt
done

for i in {1..50}
do
```

```
    rmdir toto$i
done
```

Exercice 3

Que fait le script Bash uni-ligne suivant ?

```
___$ for i in {1..100}; do echo toto$i; done
```

Exercice 4

Que font les commandes et les deux scripts Bash uni-ligne suivant ?

```
___$ mkdir test
___$ cd test
___$ for i in {1..100}; do mkdir a$i; done
___$ for i in {1..100}; do echo "ok pour a$i" > a$i/info.txt; done
```

Exercice 5

Écrire un script Bash qui crée l'arborescence suivante :

```
toto/
    -- A/
        -- info1.txt
        -- ...
        -- info10.txt
    -- B/
        -- info11.txt
        -- ...
        -- info20.txt
    -- C/
        -- info21.txt
        -- ...
        -- info30.txt
```

telle que « info<i>.txt » contient le texte « blabla i ».

TP 3 – Groupes et utilisateurs – 1

Exercice 1

a) Le fichier */etc/group* est un fichier de configuration qui permet de définir les groupes et les droits d'accès des utilisateurs.

(Le répertoire */etc* contient les fichiers de configuration).

Observez le fichier */etc/group* par la commande :

```
cat /etc/group
```

puis

```
cat /etc/group | less
```

Le fichier */etc/group* est constitué d'un ensemble de lignes qui respectent la syntaxe suivante :

```
groupe : champ_special : numero : utilisateur_1, utilisateur_n ...
```

Une ligne de ce fichier comporte les champs suivants, séparés par des caractères « : » :

1. Le nom du groupe.
2. Le mot de passe encrypté du groupe. Si ce champ est vide (presque toujours), aucun mot de passe n'est nécessaire.
3. numéro du groupe (gid).
4. liste des utilisateurs appartenant au groupe (séparés par des virgules).

b) Créez un groupe utilisateur « toto » par la commande :

```
groupadd toto
```

Si la commande est non définie faites :

```
find / -name "groupadd"
```

et résoudre le problème.

c) Contrôlez que le groupe « toto » a bien été créé par la commande :

```
cat /etc/group
```

ou :

```
tail /etc/group
```

pour n'afficher que les 10 dernières lignes du fichier */etc/group*.

ou encore :

```
cat /etc/group | grep ^toto
```

pour n'afficher que la ligne correspondante au groupe « toto ».

d) Ajoutez deux utilisateurs « util1 » et « util2 » au groupe « toto » par les commandes :

```
useradd util1 -g toto
useradd util2 -g toto
```

e) Le fichier */etc/passwd* contient les informations relatives aux utilisateurs. Chaque entrée du fichier est composée de 7 champs (ou colonnes):

1. nom de connexion (login) ;
2. mot de passe optionnel (ou x pour utiliser les mots de passes masquées (*shadow*)) ;
3. identifiant d'utilisateur (UID) ;
4. identifiant de groupe (GID) ;
5. informations libres ;
6. répertoire utilisateur ;
7. interpréteur de commande (shell) par défaut ;
8. Les champs sont séparés par le caractère « double point » (:).

Contrôlez la création des deux comptes « util1 » et « util2 ».

f) Affichez les seules lignes concernant « util1 » et « util2 » dans le fichier */etc/passwd* par :

```
cat /etc/passwd | grep ^util
```

Pourquoi ces deux lignes nous assurent que les utilisateurs « util1 » et « util2 » appartiennent bien au groupe « toto » ?

Exercice 2

(Suite de l'Exercice 1)

a) Installez le paquet `members`, qui permet d'afficher les membres d'un groupe, par la commande :

```
apt-get install members
```

b) Affichez les utilisateurs qui appartiennent au groupe « toto » par la commande :

```
members toto
```

c) Supprimez l'utilisateur « util2 » par la commande :

```
userdel util2
```

d) Contrôler la bonne suppression de « util2 » par :

```
cat /etc/passwd
```

ainsi que :

```
cat /etc/passwd | grep ^util2
```

e) De nouveau affichez les utilisateurs appartenant au groupe « toto » via la commande : `members`.

Exercice 3

(Suite des Exercices 1 et 2)

a) Considérez la ligne du fichier */etc/passwd* relative à l'utilisateur « util1 » :

```
cat /etc/passwd | grep ^util
```

b) Définir un mot de passe par défaut pour l'utilisateur « util1 » :

```
passwd util1
```

c) Connectez-vous en tant que « util1 » :

```
su util1
```

d) En tant qu' « util1 », modifiez votre mot de passe par la commande :

```
passwd
```

e) Tapez la commande ci-dessous

```
groupdel toto
```

Elle est utilisée pour supprimer le groupe « toto » ; cependant, l'opération demandée n'est pas effectuée. Pourquoi ?

f) Supprimez le groupe « toto » ainsi que l'utilisateur « util1 ». Dans quel ordre doit s'effectuer cette suppression ?

g) Assurez-vous de la bonne suppression du groupe « toto » et de l'utilisateur « util1 » par considération des fichiers */etc/group* et */etc/passwd*.

Exercice 4

a) Créez un groupe « joueurs ». Vérifiez la création de ce groupe en affichant le fichier */etc/group*.

La commande :

```
members joueurs
```

vous donne les utilisateurs du groupe « joueurs ».

b) Ajoutez les utilisateurs : « will », « max », « soso », « ikar » et « gigi » au groupe « joueurs ».

c) Vérifiez la bonne création des utilisateurs en affichant le fichier */etc/passwd*.

d) En utilisant la commande :

```
cat /etc/group | grep ^joueurs
```

Assurez-vous que les utilisateurs créés appartiennent bien au groupe « joueurs ».

e) Tapez :

```
members joueurs
```

pour connaître les membres du groupe « joueurs ».

f) Initialisez les mots de passe des utilisateurs du groupe « joueurs ».

g) Trouvez le.s groupe.s de « ikar » par la commande :

```
groups ikar
```

h) Supprimez les utilisateurs « will » et « gigi ».

   a) Constatez leurs suppressions en affichant le fichier */etc/passwd* ;

   b) Puis tapez :

```
members joueurs
```

i) Ajoutez les utilisateurs « tim » et « tina » sans spécifier leur groupe, via les commandes :

```
useradd tim
useradd tina
```

j) Assurez-vous de la bonne création de « tim » et de « tina » en affichant le fichier */etc/passwd*.

k) À quels groupes appartiennent « tim » et « tina » ?

l) Affichez le fichier */etc/group*. Que remarquez-vous ?

m) Supprimez les utilisateurs « tim » et « tina ».

n) Assurez-vous de la bonne suppression des utilisateurs « tim » et « tina » par considération des fichiers */etc/passwd* et */etc/group*.

o) Ajoutez les utilisateurs « jean » et « corinne » de groupe « joueurs », via les commandes :

```
useradd jean -g joueurs
useradd corinne -g joueurs
```

p) Affichez les membres du groupe « joueurs » en utilisant la commande `members`.

q) Supprimez les utilisateurs « max », « soso » et « ikar ».

r) Assurez-vous de la bonne suppression de ces utilisateurs par considération du fichier */etc/passwd*.

s) Que donne la commande :

```
members joueurs
```

t) Supprimez les utilisateurs « jean » et « corinne ».

u) Assurez-vous de la bonne suppression de ces utilisateurs par considération du fichier */etc/passwd*.

v) Que donne la commande :

```
members joueurs
```

w) Supprimez le groupe « joueurs ».

x) Assurez-vous de la bonne suppression du groupe « joueur » par la commande :

```
cat /etc/group | grep ^joueurs
```

TP 4 – Groupes et utilisateurs – 2

Exercice 1

a) Créez un groupe « bureau ».

b) Vérifiez la création de ce groupe en affichant le fichier */etc/group*.

c) Tapez la commande :

```
members bureau
```

qui vous donne les utilisateurs du groupe « bureau ».

d) Ajoutez les utilisateurs : « marie », « martine », « paul » et « pierre » au groupe « bureau ».

e) Vérifiez la bonne création des utilisateurs par considération des fichiers */etc/passwd* et */etc/group*.

f) Tapez :

```
members bureau
```

pour connaître les membres du groupe « bureau ».

g) Initialisez le mot de passe de « pierre ».

h) Connectez-vous en tant que « pierre », changez votre mot de passe, puis déconnectez-vous.

i) Connectez-vous en *root*, « pierre » vous informe qu'il a perdu son mot de passe. Réinitialisez le mot de passe de « pierre ».

j) Connectez-vous en tant que « pierre », changez de nouveau votre mot de passe puis déconnectez-vous.

k) Supprimez tous les utilisateurs ayant pour groupe « bureau ».

l) Assurez-vous des suppressions demandées par considération du fichier */etc/passwd* ; ainsi que de la commande `members`.

m) Supprimez le groupe « bureau ».

n) Assurez-vous de la suppression demandée par considération du fichier */etc/group*.

Exercice 2

a) Créer deux groupes « equipe_a » et « equipe_b ».

b) Créer :

- les utilisateurs « pim », « pam » et « poum » de groupe « equipe_a » ;
- les utilisateurs « zim », « zam » et « zoum » de groupe « equipe_b ».

c) Affichez les utilisateurs du groupe « equipe_a » par la commande :

```
members equipe_a
```

Faites de même pour afficher les utilisateurs du groupe « equipe_b ».

d) Supprimez l'utilisateur « poum ».

e) Affichez les utilisateurs du groupe « equipe_a ».

f) Changez l'utilisateur « zoum » de groupe en le plaçant dans le groupe « equipe_a » par la commande :

```
usermod -g equipe_a zoum
```

g) Affichez les utilisateurs du groupe « equipe_a ». Puis de même, affichez les utilisateurs du groupe « equipe_b ».

h) Supprimez les utilisateurs ayant pour groupes « equipe_a » ou « equipe_b ».

i) Assurez-vous des suppressions demandées par considération du fichier */etc/passwd*.

j) Par la commande :

```
members
```

Vérifiez que les groupes « equipe_a » et « equipe_b » ne possèdent pas de membres.

k) Supprimez les groupes « equipe_a » et « equipe_b ».

l) Assurez-vous de la suppression demandée par considération du fichier */etc/group*.

Exercice 3 [boucle « for » et ajout automatique de compte]

a) Faire :

```
touch a b c
```

Puis :

```
ls
```

b) Que fait le code Bash suivant :

```
for i in *; do echo $i; done
```

c) Créer un fichier « utilisateurs.txt » contenant des logins les uns en dessous des autres comme suit :

```
---------------------- utilisateurs.txt ----------------------
paul
marie
max
martine
toto
gogo
gigi
------------------------------------------------------------
```

d) Créer un groupe « employes ».

e) Que fait le code Bash suivant ? (le fichier « utilisateurs.txt » doit se trouver dans le répertoire de travail).

```
for lgn in $(cat utilisateurs.txt); do echo $lgn; done
```

f) La commande Bash suivante créé les utilisateurs ayant pour login les noms inscris dans le fichier « utilisateurs.txt », en leurs donnant le groupe « employes » :

```
for lgn in $(cat utilisateurs.txt); do useradd $lgn -g employes; done
```

g) Affichez le fichier */etc/passwd* pour vérifier la bonne création des utilisateurs.

h) Par la commande :

```
members
```

Affichez les utilisateurs du groupe « employes ».

i) La commande Bash suivante supprime les utilisateurs ayant pour login les noms inscris dans le fichier « utilisateurs.txt » :

```
for lgn in $(cat utilisateurs.txt); do userdel $lgn; done
```

j) Affichez le fichier */etc/passwd* pour vérifier la bonne suppression des utilisateurs.

k) Par la commande :

```
members
```

Vérifiez que le groupe « employes » ne possède pas de membres.

l) Supprimez le groupe « employes ».

m) Assurez-vous de la suppression demandée par considération du fichier */etc/group*.

TP 5 – Gestion des permissions – 1

Lecture des droits d'accès

Les droits d'accès apparaissent comme une liste de 10 symboles.

Par exemple :

```
drwxr-xr-x
```

Il faut lire ici 4 groupes de symboles comme suit :

```
d          rwx          r-x          r-x
```

Exercice 1 [Concernant le 1er symbole]

a) Créer l'arborescence suivante :

```
exercice/
├── toto
└── toto.txt
```

où :

- « toto » est un dossier ;
- « toto.txt » est un « simple » fichier.

b) Par « tree » vérifiez que vous obtenez la bonne arborescence.

c) Dans le répertoire exercice tapez :

```
ls
```

puis :

```
ls -l
```

d) Par la commande `ls -l` les droits d'accès apparaissent. Nous nous intéressons ici au premier symbole :

- `d` signifie dossier (*directory*) ;
- `-` signifie fichier.

Assurez-vous de l'exactitude ce codage lorsque vous tapez :

```
ls -l
```

Exercice 2 [Concernant le 1er symbole]

Nous nous intéressons ici au compilateur gcc pour le langage C.

a) Trouvez où se situe gcc par la commande :

which gcc

b) Normalement gcc se trouve dans */usr/bin*

Listons le contenu de */usr/bin* par la commande :

```
ls /usr/bin
```

c) L'option « `-l` » de `ls` donne les informations sur les droits d'accès.

Affichez le contenu de */usr/bin* en affichant les informations de droit d'accès.

d) La commande suivante nous permet de localiser plus précisément le compilateur gcc :

```
ls -l /usr/bin | grep gcc
```

e) Concernant gcc le premier symbole de droit d'accès est un `l` ce qui signifie que gcc est en fait un lien symbolique. Vers quel fichier pointe gcc ? De quel type est ce fichier ?

Documentation : les droits d’accès

Rappelons la manière dont les droits d’accès apparaissent comme liste de 10 symboles .

Reprenons notre exemple :

```
drwxr-xr-x
```

Qu’il faut lire comme 4 groupes de symboles comme ci-dessous :

```
d          rwx          r-x          r-x
```

Le premier groupe contient un seul symbole :

- `-` pour un fichier ;
- `d` pour un répertoire :
- `l` pour un lien.

Suivent ensuite 3 groupes de 3 symboles chacun :

<droits du propriétaire>	<droits du groupe>	<droits du reste des utilisateurs>
`rwx`	`r-x`	`r-x`

Les permissions peuvent être :

- `r` pour « permission en lecture » (*read*) ;
- `w` pour « permission en écriture » (*write*) ;
- `x` pour « permission d’exécution » pour un fichier et « permission d’entrer dans le répertoire » sinon.

Notre exemple :

```
drwxr-xr-x
```

se traduit de la manière suivante :

- `d` → c'est un répertoire ;

- `rwx` → son propriétaire peut lire (r), écrire (w) et exécuter (x) (c-à-d. entrer dans le répertoire) ;
- `r-x` → le groupe peut lire (r) et exécuter (x) (sans pouvoir modifier) ;
- `r-x` → tous les autres utilisateurs peuvent lire (r) et exécuter (x) (sans pouvoir modifier).

TP 6 – Gestion des permissions – 2

Exercice 1

a) Créer l'arborescence suivante :

```
toto/
├── 1.txt
└── tata
```

où :

« tata » est un dossier ;

« 1.txt » est un « simple » fichier.

b) Par la commande « `ls -l` » affichez les droits d'accès du répertoire toto.

c) Utiliser la commande « `ls -l | grep toto` » pour afficher la ligne nous intéresse.

Vous devez obtenir quelque chose du genre :

```
drwxrwxr-x 3 user user 4096 févr. 18 08:42 toto
```

- drwxrwxr-x → mode et droits d'accès ;
- 3 → nombre de liens physiques *;*
- user → propriétaire ;
- user → groupe ;
- 4096 → taille en octet ;
- févr. 18 08:42 → date et heure de dernière modification ;
- toto → nom.

d) Assurez-vous que toto est un répertoire.

e) En *root* créer un groupe « groupe2 ».

f) En *root* créer un utilisateur « user2 » de groupe « groupe2 ». Initialisez son mot de passe.

g) Connectez-vous en tant que « user2 » via la commande « `su` » (*switch user*) par :

```
su user2
```

- Pouvez-vous entrer dans le répertoire « toto »?
- Pouvez-vous lire le contenu du répertoire « toto » ?
- Pouvez-vous modifier le fichier « 1.txt » ?

Exercice 2 [droits pour les autres utilisateurs]

(suite de l'Exercice 1)

Soit « user » le login de l'utilisateur courant, propriétaire de l'arborescence « toto ».

a) Par la commande : « `ls -l | grep toto` » remémorez-vous les droits d'accès du répertoire « toto ».

b) En tant que « user » supprimez le droit en lecture du répertoire toto pour les « autres utilisateurs » par la commande :

```
chmod o-r toto
```

- `o` pour autre (*other*) ;
- `-r` pour supprimer la lecture (*read*).

Affichez les droits d'accès concernant le répertoire toto par la commande :

```
ls -l | grep toto
```

c) Connectez-vous en tant que « user2 ».

- Pouvez-vous entrer dans le répertoire « toto » ?
- Pouvez-vous lire le contenu du répertoire « toto » ?

d) En tant que « user » supprimez le droit en exécution du répertoire « toto » pour les « autres utilisateurs » par la commande :

```
chmod o-x toto
```

- `o` pour autre (*other*) ;
- `-x` pour supprimer l'exécution (*eXecute*).

e) Affichez les droits d'accès concernant le répertoire « toto » par la commande :

```
ls -l | grep toto$
```

f) Connectez-vous en tant que « user2 ».

- Pouvez-vous entrer dans le répertoire « toto » ?
- Pouvez-vous lire le contenu du répertoire « toto » ?

g) En tant que « user » ajouter le droit de lecture au répertoire « toto » pour les « autres utilisateurs » par la commande :

```
chmod o+r toto
```

- `o` pour autre (other)
- `+r` pour ajouter la lecture (read)

Affichez les droits d'accès concernant le répertoire toto par la commande :

```
ls -l | grep toto$
```

h) Connectez-vous en tant que « user2 ».

- Pouvez-vous entrer dans le répertoire « toto » ?
- Pouvez-vous lire le contenu du répertoire « toto » ?

Exercice 3 [groupe]

(suite des exercices 1 et 2)

Rappel : l'utilisateur courant « user » a crée l'arborescence suivante :

```
toto/
├── 1.txt
└── tata
```

où :

- « tata » est un dossier ;
- « 1.txt » est un « simple » fichier.

Nous avons également créé un groupe « groupe2 » ainsi qu'un utilisateur « user2 » de groupe « groupe2 ».

a) Visualisez les groupes de « user » et « user2 » via le fichier */etc/group*.

b) Créez un groupe « groupe3 ».

c) Ajoutez le groupe « groupe3 » aux utilisateurs « user » et « user2 » par les commandes :

```
usermod -a -G groupe3 user
```

et

```
usermod -a -G groupe3 user2
```

d) Visualisez le groupe « groupe3 » via le fichier /etc/group.

e) Utilisez la commande :

```
members groupe3
```

pour connaître les membres du groupe « groupe3 ».

f) Par la commande :

```
ls -l | grep toto$
```

trouvez le groupe du répertoire « toto » ?

g) Changez le groupe du répertoire « toto », de sorte que son groupe soit désormais le groupe « groupe3 » par la commande :

```
chgrp groupe3 toto
```

h) Visualisez la modification effectuée par la commande :

```
ls -l | grep toto$
```

Exercice 4 [droits pour le groupe]

(suite des exercices 1 → 3)

a) Listez les droits d'accès du répertoire toto par la commande ;

```
ls -l | grep toto$
```

Assurez-vous que :

- le groupe de « toto » est « groupe3 » ;
- le propriétaire de « toto » est « user » ;
- la ligne de droits d'accès de « toto » est : `drwxrwxr--`

b) En tant que « user ».

- Pouvez-vous entrer dans le répertoire « toto » ?

- Pouvez-vous lire le contenu du répertoire « toto » ?

c) Même question que précédemment pour « user2 ».

d) Les droits d'accès de « toto » sont : `drwxrwxr--`
   i. Le groupe peut-il modifier « toto » ?
   ii. Si oui, en tant que « user2 », modifiez le fichier « 1.txt ».

e) Pourquoi « user2 » ne peut pas modifier le fichier « 1.txt » ?

Exercice 5 [droits pour le propriétaire]

(suite des exercices 1 → 4)

Rappel : l'utilisateur courant « user » a crée l'arborescence suivante :

```
toto/
├── 1.txt
└── tata
```

où :

- « tata » est un dossier ;
- « 1.txt » est un « simple » fichier.

1. Nous avons créé un groupe « groupe2 ».
2. Nous avons créé un utilisateur « user2 » de groupe « groupe2 ».
3. Nous avons créé un groupe « groupe3 ».
4. Nous avons ajouté le groupe « groupe3 » aux utilisateurs « user » et « user2 ».

5. Après modification, le groupe de « toto » est désormais le groupe « groupe3 ».
6. Nous avons supprimé le droit en exécution du répertoire « toto » pour les « autres utilisateurs ».
7. De plus, le répertoire « toto » possède le droit en lecture pour les « autres utilisateurs ».

a) En affichant les droits d'accès de « toto » par la commande : « `ls -l | grep toto$` ».

Assurez-vous des points 5) 6) et 7) ci-dessus.

b) Créer un utilisateur « user3 » puis initialisez son mot de passe.

Quel est son groupe ?

c) Connectez-vous en tant que « user3 ».

- Pouvez-vous entrer dans le répertoire « toto » ?
- Pouvez-vous lire le contenu du répertoire « toto » ?

d) Retirer le droit « en exécution » pour le propriétaire par la commande :

```
chmod u-x toto
```

(« u » pour « utilisateur » ou « *user* »).

e) Assurez-vous que « toto » a le droit « en exécution » pour le groupe.

f) En tant que « user2 », pouvez-vous entrer dans le répertoire « toto » ?

g) De même, en tant que « user ». Si non, pourquoi ?

Considérer l'extrait ci-après de https://en.wikipedia.org/wiki/File_system_permissions :

> Classes
>
> Files and directories are owned by a user. The owner determines the file's *user class*. Distinct permissions apply to the owner.

> Files and directories are assigned a group, which define the file's *group class.* Distinct permissions apply to members of the file's group. The owner may be a member of the file's group.
>
> Users who are not the owner, nor a member of the group, comprise a file's *others class*. Distinct permissions apply to others.
>
> The *effective permissions* are determined based on the first class the user falls within in the order of user, group then others. For example, the user who is the owner of the file will have the permissions given to the user class regardless of the permissions assigned to the group class or others class.

Exercice 6 [droits pour le propriétaire]

(suite des exercices 1 → 5)

a) En tant que « user » ajouter les droits « en exécution » pour le répertoire « toto » pour le propriétaire, par la commande :

```
chmod u+x toto
```

b) En tant que « user » :

- enlever les droits de lecture et d'écriture pour le fichier « 1.txt » pour « les autres » ;
- ajouter les droits de lecture et d'écriture pour le fichier « 1.txt » pour le groupe.

c) Assurez-vous que « user2 » ne puisse ni lire ni modifier le fichier « 1.txt » en vous connectant sous « user2 », puis en testant, par exemple, les commandes suivantes :

- `cat 1.txt`
- `echo "blabla" >> 1.txt`
- `vi 1.txt`

d) Assurez-vous que « user » puisse lire et modifier le fichier « 1.txt ».

e) « user » peut lire et modifier « toto/1.txt » contrairement à « user2 ».

Pourtant :

- « `ls -l | grep toto$` » indique que le groupe de toto est « groupe3 » ;
- « `groups user2` » indique que « groupe3 » est un groupe de « user2 » ;

- de plus, en b) nous avons ajouté les droits en lecture et écriture pour « toto/1.txt » pour le groupe.

Donnez l’explication.

TP 7 – Scripts Bash, Utilisateurs et Permissions

Assurez-vous du bon fonctionnement de vos scripts via :

- le fichier */etc/passwd* ;
- le fichier */etc/group* ;
- la commande `tail` ;
- la commande `cat` ;
- la commande `grep` ;
- la commande `members` ;
- la commande `groups` ;
- et la commande `ls -l`

Exercice 1

Écrire un script Bash qui crée :

1. un groupe « toto » ;
2. les utilisateurs u1, …, u30 de groupe primaire « toto ».

Exercice 2

(Suite de l'Exercice 1)

Écrire un script Bash qui supprime :

1. les utilisateurs u1, …, u30 ;
2. le groupe « toto ».

Exercice 3

Écrire un script Bash qui crée :

1. un groupe « toto » ;

2. un groupe « tata » ;
3. les utilisateurs u1, …, u15 de groupe primaire « toto » ;
4. les utilisateurs u16, …, u30 de groupe primaire « tata ».

Exercice 4

(Suite de l'Exercice 3)

Écrire un script Bash qui supprime :

1. les utilisateurs u1, …, u30 ;
2. le groupe « toto » ;
3. le groupe « tata ».

Exercice 5

Écrire un script Bash qui crée :

1. l'arborescence suivante :

```
A/
-- Â1/
-- ...
-- A10/
```

2. le groupe « a » ;

3. les utilisateurs a1, …, a10 de groupe « a » ;

4. l'arborescence est telle que chaque dossier « A<i>/ » est :
   a) de propriétaire « a<i> » ;
   b) de groupe « a » ;
   c) permis en entrée, lecture et écriture à l'utilisateur « a<i> » ;
   d) permis en entrée et lecture ; et interdit en écriture aux utilisateurs de groupe « a ».
   e) interdit en entrée, lecture et écriture aux « autres utilisateurs » ;

5. Tester la bonne exécution du script.

6. Écrire un script de suppression :
   a) des utilisateurs a1, …, a10 ;
   b) du groupe « a » ;
   c) ainsi que de l’arborescence « A/ ».

TP 8 – Scripts Bash, arguments d’un script Bash

Exercice 1

Soit le script Bash suivant :

```
--- exo1.sh ---
#!/bin/bash

echo "$0"
echo "$1"
echo "$2"

echo "$#"

echo "$@"
```

Que donne :

- `bash exo1.sh`
- `bash exo1.sh toto tata`
- `bash exo1.sh toto tata bobo`
- `bash exo1.sh 1 2 10 1 2`
- `bash exo1.sh a1 a2 a3`

Exercice 2

Soit le script Bash suivant :

```
--- exo2.sh –
#!/bin/bash

echo "le script s'appelle $0"
echo "il possede $# arguments"
```

Que donne :

- `bash exo2.sh`
- `bash exo2.sh toto tata`
- `bash exo2.sh a1 a2 a3`

Exercice 3

Soit le script Bash suivant :

```
--- exo3.sh –
#!/bin/bash

if [ "$#" -lt 1 ]
then
   echo "Usage : $0 demande au moins un argument"
   exit 1
fi

echo "$1"
```

Que donne :

- `./exo3.sh`
- `./exo3.sh ; echo $?`
- `./exo3.sh toto`
- `./exo3.sh toto; echo $?`
- `./exo3.sh toto tata`
- `./exo3.sh toto tata  ; echo $?`

Exercice 4

Soit le script Bash suivant :

```
--- exo4.sh –
#!/bin/bash
```

```
for i in $(seq 10)
do
   echo "$i"
done
```

Que donne :

- `bash exo4.sh`
- `bash exo4.sh toto`

Exercice 5

Soit le script Bash suivant :

```
--- exo5.sh ---
#!/bin/bash


for i in $(seq $1)
do
   echo "$i"
done
```

Que donne :

- `bash exo5.sh 1`
- `bash exo5.sh 2`
- `bash exo5.sh 6`
- `bash exo5.sh 10`
- Pourquoi les appels ci-dessous échouent ?
- `bash exo5.sh`
- `bash exo5.sh toto`

Exercice 6

Soit le script Bash suivant :

```
--- exo6.sh ---
#!/bin/bash

if [ "$#" -lt 1 ]
then
   echo "Usage: $0 demande au moins un argument"
   exit 1
fi


for x in $@
do
   echo $x
done
```

Que donne :

- ./exo6.sh
- ./exo6.sh ; echo $?
- ./exo6.sh toto
- ./exo6.sh toto ; echo $?
- ./exo6.sh toto tata
- ./exo6.sh toto tata ; echo $?
- ./exo6.sh a1 a2 a3
- ./exo6.sh a1 a2 a3 ; echo $?

Exercice 7

Que fait le script Bash suivant :

```
--- exo7.sh ---
#!/bin/bash

echo "entrer une valeur"
read  val
```

```
echo " --> $val"
```

Exercice 8

Que fait le script Bash suivant :

```
--- exo8.sh ---
#!/bin/bash

echo -n "entrer une valeur: "
read  val
echo " --> $val"
```

Exercice 9

Écrire un script Bash *exo9.sh* qui agit comme suit :

```
___$ ./exo9.sh

Votre nom ?

Toto

Bonjour Toto

Quel âge avez-vous ?

10

C’est enregistré, vous avez 10 ans.
```

Exercice 10

Écrire un script Bash *exo10.sh* qui :

-- prends 2 arguments ;

-- affiche un *usage* pertinent dans le cas où le nombre d’arguments n’est pas égal à 2 ;

-- sinon affiche les arguments à la manière de l’exemple ci-dessous :

```
___$ ./exo10.sh toto tata

arg1 = toto et arg2 = tata
```

Exercice 11

Que fait le script Bash suivant :

```
--- exo11.sh ---
#!/bin/bash

for x in $( ls )
do
   echo $x
done
```

Exercice 12

Écrire un script Bash *exo11.sh* qui :

-- prends 1 argument ;

-- affiche un *usage* pertinent dans le cas où le nombre d'arguments n'est pas égal à 1 ;

– agit à la manière de l'exemple ci-dessous.

Exemple

```
___$ echo "1.txt" > 1.txt
___$ bash exo12.sh 1.txt
L'argument entré est un fichier
___$ mkdir toto
___$ bash exo12.sh toto
toto est un répertoire
___$ ln -s 1.txt 2.txt
___$ ./exo11.sh 2.txt
2.txt est un fichier
```

Exercice 13

Écrire un script Bash *exo13.sh* qui affiche les répertoires présents dans le répertoire courant.

TP 9 – Scripts Bash, grep, sed, wc et regex

Exercice 1 [grep]

Écrire un script Bash *exo1.sh* qui :

– prend un argument <arg> ;

– affiche un usage pertinent dans le cas où le nombre d'arguments n'est pas égal à 1 ;

– affiche les paquets Linux traitant de <arg> et dont le nom commence par <arg>.

Tester :

```
___$ bash exo1.sh vim
```

Exercice 2 [grep]

Écrire un script Bash *exo2.sh* qui :

– prend deux arguments <arg1> et <arg2> ;

– affiche un usage pertinent dans le cas où le nombre d'arguments n'est pas égal à 2 ;

– affiche les paquets Linux traitant de <arg1> et dont le nom commence par <arg2>.

Tester :

```
___$ bash exo2.sh python idle
```

Exercice 3 [sed]

Considérer le fichier *toto.txt* ci-dessous :

```
---  toto.txt ---
toto toto
tata
Tata
tototatatoto
ToTo
```

Tester les commandes ci-après.

```
___$ cat toto.txt
___$ cat toto.txt | sed s/toto/AAA/
___$ cat toto.txt
___$ cat toto.txt | sed s/toto/AAA/g
```

Note [concernant les exercices 4→9] :

Pour vos tests, créez les fichiers : *abc.txt, abctxt, toto.txt, toto.TXT., a123b.txt* et *a123btxt.*

Exercice 4 [expressions régulières]

Que font les scripts Bash ci-dessous.

```
--- exo4a.sh ---
#!/bin/bash

for x in $(ls)
do
   if [[ $x =~ \.txt$ ]]
   then
      echo $x
   fi
done
```

```
--- exo4b.sh ---
#!/bin/bash

for x in $(ls *.txt)
do
   echo $x
done
```

Exercice 5 [expressions régulières]

Que font les scripts Bash ci-dessous.

```
--- exo5a.sh ---
```

```
#!/bin/bash

for x in $(ls)
do
   if [[ $x =~ ^a ]]
   then
      if [[ $x =~ \.txt$ ]]
      then
         echo $x
      fi
   fi
done
```

```
--- exo5b.sh ---
#!/bin/bash

for x in $(ls a*.txt )
do
   echo $x
done
```

Exercice 6 [expressions régulières]

Que font les scripts Bash ci-dessous.

```
--- exo6a.sh ---
#!/bin/bash

for x in $(ls)
do
   if [[ $x =~ [0-9] ]]
   then
      echo $x
   fi
done
```

```
--- exo6b.sh ---
#!/bin/bash

for x in $(ls *[0-9]*)
do
   echo $x
```

```
done
```

Exercice 7

Écrire un script Bash *exo7.sh* qui :

– prend en argument un répertoire <arg> ;

– affiche un usage pertinent dans le cas où le nombre d'arguments n'est pas égal à 1 ;

– affiche un usage pertinent dans le cas où l'argument <arg> n'est pas un répertoire ;

– affiche les fichiers (en Unix tout est fichier) présents dans <arg>, qui :

  – commence par « a » ;

  – se terminent par « .txt » ;

  – contiennent au moins 1 chiffre.

Exercice 8

Que font les scripts Bash suivant :

```
--- exo8a.sh ---
#!/bin/bash

for x in $(ls)
do
   if [[ $x =~ \.TXT$ ]]
   then
      echo $x
      echo $( echo $x | sed "s/\.TXT$/\.txt/" )
   fi
done
```

```
--- exo8b.sh ---
#!/bin/bash

for x in $(ls *.TXT)
do
   echo $x
   echo $( echo $x | sed "s/\.TXT$/\.txt/" )
```

```
done
```

Exercice 9

Écrire un script Bash *exo9.sh* qui remplace, dans le répertoire courant, les noms de fichiers d'extension « .TXT » en « .txt ».

Exercice 10 [wc]

Écrire un script Bash *exo10.sh* qui affiche le nombre de liens symboliques présents dans le répertoire courant.

Exercice 11 [wc]

Écrire un script Bash *exo11.sh* qui affiche le nombre de fichiers cachés présents dans le répertoire courant.

TP 10 – Scripts Bash, grep, find et awk

Exercice 1 [grep]

Créer l'arborescence ci-après :

```
titi
├── tata.txt
├── toto
│   └── titi.txt
└── toto.txt
```

telle que :

-- « titi » et « toto » sont des répertoires ;

-- t*ata.txt* contient le texte : « ToTo tata » ;

-- *toto.txt* contient le texte : « toto » ;

-- *titi/titi.txt* contient le texte : « tata toto toto ».

Nous supposons que « titi » se trouve dans le répertoire ~ ; et que l'utilisateur se situe dans ce répertoire.

1. Que font les commandes :

```
ls | grep toto
ls | grep Toto
ls | grep -i Toto
```

Nous nous plaçons au sein du répertoire « titi ».

2. Que font les commandes :

```
grep "toto" $(ls)
grep -i "toto" $(ls)
```

3. Que font les commandes :

```
grep -r "toto"
```

```
grep -ir "toto"
```

Nous nous plaçons au sein du répertoire « ~ », dans lequel est supposé se trouver le répertoire « titi » .

4. Que font les commandes :

```
grep  "toto" titi
grep -r "toto" titi
grep -ir "toto" titi
grep -ir "^toto" titi
grep -ir "toto$" titi
```

Nous nous plaçons au sein du répertoire « titi ».

5. Que font les commandes :

```
ls | grep ".txt$"
ls | grep ".txt$" | wc -l
grep -r "toto" .
grep -r "toto" . | wc -l
grep -r "^toto" .
grep -r "^toto" . | wc -l
```

Exercice 2 [grep]

Écrire un script Bash : *super_grep.sh* qui :

-- prend deux arguments : <arg1> et <arg2> ;

-- affiche un usage pertinent dans le cas où le nombre d'arguments n'est pas égal à 2 ;

-- affiche un usage pertinent dans le cas où <arg1> n'est pas un répertoire ;

-- sinon affiche, sans tenir compte de la casse, le résultat de la recherche de <arg2> au sein du contenu des fichiers présents dans l'arborescence de racine <arg1>.

Exercice 3 [find]

Que font les commandes ci-après ?

```
find titi -iname toto
find titi -iname "*toto*"
find titi -iname "*toto*" -type f
find titi -iname "*toto*" -type d
find titi -iname "*.txt"
```

Exercice 4 [find]

Écrire un script Bash : *find2.sh* qui :

-- prend deux arguments : <arg1> et <arg2> ;

-- affiche un usage pertinent dans le cas où le nombre d'arguments n'est pas égal à 2 ;

-- affiche un usage pertinent dans le cas où <arg1> n'est pas un répertoire ;

-- sinon affiche le résultat de la recherche :

```
find <arg1> -iname <arg2>
```

Exercice 5 [find]

Écrire un script Bash : *find3.sh* qui procède comme précédemment, mais créer un fichier :

resultats__<arg1>_<arg2>__.txt

contenant les résultats de la recherche, au lieu d'afficher les résultats dans le terminal.

Exercice 6 [find]

Écrire un script Bash : *cpt_txt.sh* qui :

-- prend un argument : <arg> ;

-- affiche un usage pertinent dans le cas où le nombre d'arguments n'est pas égal à 1 ;

-- affiche un usage pertinent dans le cas où <arg> n'est pas un répertoire ;

-- sinon affiche le nombre fichiers « .txt » présent dans l'arborescence de racine <arg>.

Exercice 7 [awk]

Considérer le fichier suivant.

```
--- gogo.txt ----------
a b c d 1
aa bb cc dd 2
aaa bbb ccc ddd 3
aaaa bbbb cccc dddd 4
----------------------
```

1/ Que donnent les commandes suivantes :

```
cat gogo.txt

cat gogo.txt | awk '{ print $1 }'

cat gogo.txt | awk '{ print $2 }'

cat gogo.txt | awk '{ print $3 }'

cat gogo.txt | awk '{ print $4 }'

cat gogo.txt | awk '{ print $5 }'

cat gogo.txt | awk '{ print $0 }'

cat gogo.txt | awk '{ print $6 }'

cat gogo.txt | awk '{ print $1 $5 }'

cat gogo.txt | awk '{ print $1 " " $5 }'

cat gogo.txt | awk '{ print $5 " " $1 }'
```

2/ Que donnent les commandes suivantes :

```
ls -l

ls -l | awk '{ print $1 }'

ls -l | awk '{ print $4 }'

ls -l | awk '{ print $6 }'

ls -l | awk '{ print $8 }'

ls -l | awk '{ print $9 }'
```

```
ls -l | awk '{ print $0 }'
```

3/ Que donnent les commandes suivantes :

```
ls -l | grep -v "^total" | awk '{ print $1 }'
```

Exercice 8 [awk]

Écrire un script Bash : *proprio.sh* qui :

-- prend en argument un nom de fichier <arg> ; (rappel : en Unix tout est fichier) ;

-- affiche un usage pertinent dans le cas où le nombre d'argument n'est pas égal à 1 ;

-- affiche un usage pertinent dans le cas où le fichier donné en argument n'est pas présent dans le répertoire courant ;

-- sinon affiche le propriétaire de <arg>.

Exercice 9 [awk]

Écrire un script Bash : *ls_symb.sh* qui affiche les liens symboliques présents dans le répertoire courant.

Exercice 10 [awk]

Écrire un script Bash : *cpt_symb.sh* qui affiche le nombre de liens symboliques présents dans le répertoire courant.

Exercice 11 [awk]

Écrire un script Bash : *ls_pg.sh* qui affiche, pour chaque fichier, son propriétaire et son groupe.