TP OS n°4

Sommaire

Les objectifs de ce tp sont d'être capable, en utilisant des commandes de base sous GNU/-Linux, de gérer les droits de base sur les fichiers et les répertoires et d'assurer des tâches simples d'administration.

Introduction

Environnement de travail

Il vous faut ouvrir une **session** sur votre poste de travail. Vous pouvez utiliser soit le mode console soit l'interface graphique. Dans les deux cas, vous allez travailler **« en ligne de commande »**.

Pour ce TP, il vous faudra préalablement créer un répertoire de travail `tpos4` dans `$HOME/tpos` en tapant la commande suivante :

```
$ mkdir -p $HOME/tpos/tpos4
```

Pour se déplacer dans l'arborescence de travail, il faut taper la commande suivante :

```
$ cd $HOME/tpos/tpos4
```

Les droits sur les fichiers

Chaque fichier du système est associé à des droits d'accès. Les droits de base sur les fichiers sont :
– `r` (*read*) : droit de **lire** le fichier
– `w` (*write*) : droit d'**écrire** dans le fichier
– `x` (*execute*) : droit d'**exécuter** le fichier (programme ou script)

Remarque : certains de ces droits ont un sens différent pour les répertoires (voir plus loin).

Ceux-ci sont affichés par la commande `ls` en utilisant le format long : `ls -l`. Les droits seront affichés dans l'ordre `ugo` : (Si l'un des droits n'est pas accordé, un tiret est affiché à sa place)

		droits concernant le propriétaire du fichier			droits concernant les autres membres du même groupe que le groupe propriétaire du fichier			droits concernant tous les autres utilisateurs		
$ ls -l	type de fichier	u(ser)			g(roup)			o(ther)		
-rwxrw-r-- unfichier	-	r	w	x	r	w	-	r	-	-
	indique un fichier ordinaire	Le propriétaire peut lire, écrire et exécuter ce fichier			Les membres du même groupe propriétaire peuvent lire et écrire, mais pas exécuter ce fichier			Les autres utilisateurs peuvent seulement lire ce fichier		

Décomposition de l'affichage avec ls -l

Pour gérer ces droits, on peut utiliser soit :
– le mode **littéral** : les catégories `ugo` et `a` (*all*, pour tous), les caractères `rwx`
– le mode **numérique** (en octal) : `r` ($2^2 = 4$) + `w` ($2^1 = 2$) + `x` ($2^0 = 1$) soit par exemple ($644 \rightarrow rw-r--r--$)

$ ls -l

-rwxrw-r-- unfichier

	u			g			o		
	r	**w**	**x**	**r**	**w**	**-**	**r**	**-**	**-**
	2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
	4	2	1	4	2	0	4	0	0
En octal ->	7			6			4		

En plus de ces droits de base, il existe aussi pour les **fichiers** :
– le droit `s` (bloc `u`) : utilise l'UID (identifiant) du propriétaire (Set-UID) lors de l'exécution du fichier à la place l'UID de l'utilisateur
– le droit `s` (bloc `g`) : utilise l'ID (identifiant) du groupe propriétaire (Set-GID) lors de l'exécution du fichier
– le droit `t` (bloc `o`) : pour la conservation du code en mémoire lors de l'arrêt de l'exécution

On utilise la commande `chmod` pour modifier les droits sur un fichier :

```
$ man chmod
SYNOPSIS : chmod [options] mode fichier ...
```

Remarque : une option utile est `-R` *qui permet de modifier récursivement les autorisations des répertoires et de leurs contenus.*

Pour ajouter (+) ou enlever(-) un (ou plusieurs) droit, on peut utiliser soit le mode littéral soit le mode numérique. Pour la valeur du mode, on peut fournir 3 ou 4 chiffres (le premier chiffre étant facultatif) : le premier chiffre (facultatif) correspond au droit `s` ou `t`, le deuxième chiffre à `u`, le troisième à `g` et le quatrième à `o`.

```
$ chmod 666 unfichier  : ce qui correspond à rw-rw-rw-
$ chmod 2666 unfichier : idem avec en plus le droit s pour g (Set-GID)
```

Remarque : attention, la vérification des droits d'accès se fait dans l'ordre `ugo`*. Dès qu'une concordance est trouvée, elle s'applique.*

L'affichage des droits `s` ou `t` est particulier car aucun emplacement a été ajouté et il se fait en utilisant une minuscule ou une majuscule comme ceci :

```
$ ls -l
// droit s pour u
-rwSr--r-- 1 tv tv 0 sep 19 10:09 unfichier (sans le droit x)
-rwsr--r-- 1 tv tv 0 sep 19 10:09 unfichier (avec le droit x)

// droit s pour g
-rw-r-Sr-- 1 tv tv 0 sep 19 10:09 unfichier (sans le droit x)
-rw-r-sr-- 1 tv tv 0 sep 19 10:09 unfichier (avec le droit x)
```



```
// droit t pour o
-rw-r--r-T 1 tv tv 0 sep 19 10:09 unfichier (sans le droit x)
-rw-r--r-t 1 tv tv 0 sep 19 10:09 unfichier (avec le droit x)
```

Lorsqu'un nouveau fichier est créé, on distingue deux situations particulières :
– la création d'un fichier : quels sont les droits par défaut ?
– la copie d'un fichier : quels sont les droits du fichier copié ?

Les droits par défaut d'un nouveau fichier sont définis par rapport à un **masque** des droits défini pour chaque utilisateur avec la commande `umask`. La commande `umask` permet donc d'afficher ou de modifier le masque de création de fichier de l'utilisateur.

```
$ help umask
```

```
// En octal :
$ umask
0022
```

```
// En littéral :
$ umask -S
u=rwx,g=rx,o=rx
```

Un fichier est toujours créé par un programme : une commande (`touch`, `cat`, `cp`, ...), un éditeur (`vim`, `kwrite`, ...), un compilateur (`gcc`), ou tout autre application (`dolphin`, `syslog`, ...).

a) Le programme utilisé définit les droits qu'il désire pour le fichier à créer, par exemple :
   – rw-rw-rw- (666) pour des fichiers réguliers (non exécutable)
   – rwxrwxrwx (777) pour des fichiers exécutables

b) Puis le système applique le masque défini par `umask` pour créer les droits du fichier :

droits d'origine (666)	r	w	-	r	w	-	r	w	-
masque (022)	0	0	0	0	**1**	0	0	**1**	0
opération droits d'origine & ~masque	↓	↓	↓	↓	**X**	↓	↓	**X**	↓
droits du fichier nouvellement créé	r	w	-	r	-	-	r	-	-

Soit l'opération suivante : 666 & ~022 = 644 = rw- r– r–

Lors de la copie d'un fichier, c'est le même principe qui est appliqué en utilisant cette fois les **droits du fichier source**.

Remarque : par contre si le fichier destination existe (écrasement), le masque n'est pas utilisé et à la place on utilise les droits du fichier destination : droits fichier source `&` *droits fichier destination*

Il existe des options (`-p`, `-a`, ...) qui modifient ce comportement et permettent de préserver les propriétés du fichier source.

Pour **changer de propriétaire**, deux commandes permettent cette modification :
– `chown` : propriétaire (et groupe propriétaire)
– `chgrp` : groupe propriétaire

Évidemment, elles ne seront autorisées et applicables que si :
– on a les droits suffisants
– on est le propriétaire
– on est *root*

Différents types de fichiers

> « Tout est fichier »

Il en est ainsi sous Unix, et donc sous Linux : tous les éléments du système sont manipulés par des fichiers ! Donc, une image GIF est un fichier, un répertoire est un fichier, le disque est un fichier, la mémoire est un fichier ...

Toutefois cette uniformisation permet de simplifier grandement la manipulation du système en général, et sa programmation en particulier.

Pour pouvoir représenter tout cela, il existe plusieurs types de fichiers :
– **Les fichiers "normaux"** : les fichiers au sens classique du terme (identifié par un tiret `-`)

```
$ ls -l /dead.letter
-rw------- 1 root root 324099 2010-05-25 13:13 /dead.letter
$ file /dead.letter
/dead.letter: regular file, no read permission
```

– **Les répertoires** : un répertoire (identifié par un `d`) est un fichier contenant des références et des descriptions pour d'autres fichiers (qui peuvent être des répertoires...).

```
$ ls -ld /tmp/
drwxrwxrwt 24 root root 4096 2010-07-31 15:44 /tmp/
$ file /tmp
/tmp: directory
```

– **Les fichiers "spéciaux"** : ces fichiers, stockés dans `/dev`, permettent l'accès aux entrées/sorties d'un périphérique. Chacun d'entre eux est identifié par un nom et surtout par deux numéros uniques : majeur et mineur. On distingue deux types (identifié par un `c` ou un `b`) : caractères (terminal, port série, imprimante, etc) ou blocs (disque, mémoire, etc).

```
$ ls -l /dev/null
crw-rw-rw- 1 root root 1, 3 2010-07-29 09:40 /dev/null
$ file /dev/null
/dev/null: character special

$ ls -l /dev/sda
brw-rw---- 1 root disk 8, 0 2010-07-29 09:40 /dev/sda
$ file /dev/sda
/dev/sda: block special
```

– **Les liens symboliques** : un lien symbolique est un *alias* d'un fichier ou d'un répertoire (identifié par un `l`)

```
$ ls -ld /etc/init.d
lrwxrwxrwx 1 root root 11 2010-02-01 12:50 /etc/init.d -> rc.d/init.d/
$ file /etc/init.d
/etc/init.d: symbolic link to ‘rc.d/init.d’
```


– **Les fichiers "*sockets*"** : permettent des communications locales inter processus sur une machine (identifié par un `s`)

```
$ ls -l ~/tmp/ksocket-tv/kdeinit4__0
srw------- 1 tv tv 0 2010-07-29 07:41 /home/tv/tmp/ksocket-tv/kdeinit4__0
$ file ~/tmp/ksocket-tv/kdeinit4__0
/home/tv/tmp/ksocket-tv/kdeinit4__0: socket
```

Les droits sur les répertoires

Dans le cas des répertoires, l'interprétation des droits est légèrement différente de celle concernant les fichiers. Les informations concernant un répertoire sont obtenues par la commande : `ls -dl <nom du répertoire>`

```
$ ls -dl /tmp
drwxrwxrwt 8 root root 4096 2009-09-05 16:46 /tmp/
```

Remarque : le **`d`** *devant indique bien que c'est un répertoire (directory)*

```
$ file /tmp
/tmp: directory
```

L'interprétation des droits d'accès pour les répertoires est la suivante :
– `r` : autorise la **lecture** du contenu du répertoire (permet donc de voir la liste des fichiers contenus dans le répertoire)
– `w` : autorise la **création**, la **suppression** et le **changement** du nom d'un élément du répertoire. Cette permission est indépendante de l'accès aux fichiers dans le répertoire
– `x` : autorise l'**accès** au répertoire (droit de traverser et de se déplacer par exemple avec la commande `cd`)

Remarque : la permission de supprimer un fichier n'est pas attribuée au fichier lui-même, mais elle dépend uniquement du droit d'écriture dans le répertoire qui le contient.

Cela pose un problème pour des répertoires « partagés ». Par exemple pour le répertoire `/tmp`, tout utilisateur doit pouvoir créer des fichiers dans ce répertoire. Les droits pour ce répertoire seront `rwx rwx rwx`. L'ensemble des utilisateurs (type d'utilisateur `other`) pourront donc lire et écrire dans `/tmp`. Mais un utilisateur pourrait supprimer un fichier créé par un autre utilisateur !

Le bit `t` (*sticky bit*) permet de régler ce problème : un utilisateur pourra supprimer un fichier que s'il en est propriétaire. Le bit `t` apparaît à la place du bit `x` de `other`.

```
// Affichage du sticky bit :
$ ls -dl /tmp
// si le bit x est positionné :
drwxrwxrwt 8 root root 4096 2009-09-05 16:46 /tmp/
// si le bit x n'est pas positionné :
drwxrwxrwT 8 root root 4096 2009-09-05 16:46 /tmp/
```


Lors d'une session, un utilisateur appartient à son groupe de rattachement principal : celui définit dans le fichier `/etc/passwd`.

```
//Affichage des informations sur utilisateur :
$ id
$ cat /etc/passwd | grep $USER
$ getent passwd | grep $USER
tv:x:500:500:Vaira Thierry:/home/tv:/bin/bash

// l'utilisateur tv a pour groupe principal le GID 500
```

Cet utilisateur peut toutefois être membre d'autres groupes : cela est défini dans le fichier `/etc/group`.

```
// Affichage des informations de groupe sur utilisateur :
$ groups
$ cat /etc/group
$ getent group
tv:x:500:
projet:x:503:tv,theo

// l'utilisateur tv a pour groupe principal tv (GID 500) et il est aussi membre du groupe
    projet (GID 503)
```

Le système d'exploitation GNU/Linux applique les comportements suivants en cas d'appartenance à plusieurs groupes :

- les permissions de groupe d'un fichier (ou d'un répertoire) sont applicables à tout utilisateur membre du groupe propriétaire du fichier
- tout fichier nouvellement créé a pour groupe propriétaire le groupe effectif de l'utilisateur qui le crée si le répertoire qui le contient n'est pas SGID
- tout fichier nouvellement créé a pour groupe propriétaire le groupe auquel appartient le répertoire dans lequel il est créé si ce répertoire est SGID. Si un répertoire est SGID, tout sous-répertoire créé ultérieurement héritera du bit SGID. Ce comportement est très adapté au travail de groupe.

Rappel : attention, la vérification des droits d'accès se fait dans l'ordre ***ugo****. Dès qu'une concordance est trouvée, elle s'applique.*

Manipulations

Étape n°1 : Les droits sur les fichiers

Se déplacer dans l'arborescence :

```
$ cd $HOME/tpos/tpos4
```

Créer un fichier source avec l'éditeur `vim` :

```
$ vim hello.c
```

Éditer le fichier `hello.c` avec le contenu ci-dessous :

```c
// Affiche à l'écran le message Hello world

#include <stdio.h>

#define TEXTE "Hello world"

int main ()
{
     printf("%s\n", TEXTE);
     return 0;
}
```

Le fichier source hello.c

Fabriquer un fichier binaire :

```
$ gcc hello.c -o hello
$ ls -hl hello
-rwxr-xr-x 1 tv tv 5,7K 2010-07-30 13:39 hello
-rw-r--r-- 1 tv tv 74 2010-07-30 13:39 hello.c
```

Exécuter un fichier binaire :

```
$ ./hello
Hello wordl
```

Retirer tous les droits à ces deux fichiers :

```
$ chmod 000 hello*   // ou
$ chmod a-rwx hello*
```

*Remarque : le caractère * est un joker qui remplace tous les caractères*

Éditer le fichier `hello.c` avec `vim` :

```
$ vim hello.c
"hello.c" [Permission refusée]
```

=> impossible vous n'avez pas le droit de lecture r

Ajouter le droit `r` pour le propriétaire du fichier `hello.c` :

```
$ chmod 400 hello.c  // ou
$ chmod u+r hello.c
```

Éditer le fichier `hello.c` avec `vim`, le modifier puis l'enregistrer :

```
$ vim hello.c
"hello.c" [lecture-seule]
=> impossible vous n'avez pas le droit d'écriture w
```

Remarque : le système vous autorise à outrepasser la protection readonly (avec `vim` *faire* `:w !`*) parce que vous êtes le propriétaire de ce fichier.*

Supprimer un fichier protégé en écriture :

```
$ rm hello.c
rm: détruire un fichier protégé en écriture fichier régulier 'hello2.c'?
=> le droit w intervient aussi pour la suppression
```

Exécuter le programme `hello` :

```
$ ./hello
bash: ./hello: Permission denied
=> impossible vous n'avez pas le droit d'exécution x
```

Ajouter le droit d'exécution au programme `hello` et l'exécuter :

```
$ chmod 100 hello   // ou
$ chmod u+x hello
$ ./hello
Hello world
=> le droit r n'est pas indispensable
```

Afficher les droits actuels :

```
$ ls -l
---x------ hello
-r-------- hello.c
```

Mettre tous les droits (sauf `x`) pour tout le monde pour `hello.c` :

```
$ chmod 666 hello.c  // ou
$ chmod ugo+rw hello.c
```

Faire une copie de sauvegarde du fichier `hello.c` :

```
$ cp hello.c hello.bak
$ ls -l
-rw-rw-rw- hello.c
-rw-r--r-- hello.bak
```

Créer le fichier `gagnerAuLoto.sh` :

```
$ vim gagnerAuLoto.sh
echo "Perdu !"
ls -l gagner*
```

Mettre le droit `x` sur le fichier `gagnerAuLoto.sh` et exécuter le :


```
$ chmod +x gagnerAuLoto.sh
$ ./gagnerAuLoto.sh
Perdu !
-rwxr-xr-x 1 tv tv 29 2010-07-30 14:00 gagnerAuLoto.sh
```

Vous venez de créer votre premier script !

Étape n°2 : Les droits sur les répertoires

Se déplacer dans l'arborescence :

```
$ cd $HOME/tpos/tpos4
```

Créer un répertoire `temp` et lister ses droits :

```
$ mkdir temp
$ ls -dl
drwxr-xr-x temp/
```

Copier des fichiers dans un répertoire :

```
$ cp hello* ./temp
```

Enlever tous les droits au répertoire `temp` :

```
$ chmod 000 ./temp   // ou
$ chmod a-rwx ./temp
```

Lister le contenu du répertoire `temp` :

```
$ ls -l ./temp
ls: temp: Permission denied
```

=> impossible il manque le droit r

Se déplacer dans `temp` :

```
$ cd temp
bash: cd: temp: Permission denied
```

=> impossible il manque le droit x

Mettre le droit `x` pour `temp` et exécuter le programme `hello` :

```
$ chmod 100 ./temp   // ou
$ chmod u+x ./temp
$ ./temp/hello
Hello World !
```

=> on peut accéder et exécuter le programme hello car on savait qu'il existait !

Se déplacer dans `temp` et lister son contenu :

```
$ cd temp
$ ls
ls: .: Permission denied
```

=> on peut accéder au répertoire mais on ne peut lister son contenu.

Mettre seulement le droit `r` à `temp` et refaire la manipulation précédente :

```
$ chmod 400 ./temp
$ ./temp/hello
bash: ./temp/system: Permission denied

$ ls -l ./temp
ls: ./temp/hello: Permission denied
ls: ./temp/hello.c: Permission denied
```

=> on peut lire le nom des fichiers mais on ne peut pas accéder aux informations sur ces fichiers !

Mettre les droits `rx` pour `tmp` et copier le fichier `/etc/passwd` dedans :

```
$ chmod 500 ./tmp
$ cp /etc/passwd ./tmp
cp: ne peut créer le fichier régulier './tmp/passwd': Permission denied
```

=> il manque le droit w pour écrire dans ce répertoire !

Mettre le droit `w` et refaire la copie :

```
$ chmod 200 temp
$ cp /etc/passwd ./temp
cp: ne peut évaluer './tmp/passwd': Permission denied
```

=> il faut au moins les droits wx pour écrire dedans ! Les droits normaux pour un propriétaire sont rwx sur un répertoire.

Mettre les droits `rwx` pour tous et faire une copie du répertoire `temp` et de son contenu :

```
$ chmod 777 temp
$ cp -R temp tmp
$ ls -l
drwxr-xr-x tmp/
drwxrwxrwx temp/
```

=> le masque fonctionne aussi pour les répertoires !

Questions de révision

L'idée de base des questions de révision est de vous donner une chance de voir si vous avez identifié et compris les points clés de ce TP.

Extrait du contenu d'un répertoire avec les différents droits :

```
drwxr-x--- 15 dupont profs 4096 Jan 1 11:20 ./
drwxr----x 83 dupont profs 4096 Jan 1 13:47 ../
-rwxrwxrwx 1 dupont prof 42 Oct 21 15:16 boucle
drwxrwxr-x 3 dupont prof 1200 Oct 24 09:54 c
drwxrwxr-x 2 dupont prof 80 Apr 16 1991  cobol
drwxr-xr-x 2 dupont prof 48 Apr 3 1991   comptes_util
drwxrwxr-x 4 dupont prof 336  Oct 24 15:32  courrier
drwxrwxr-x 2 dupont prof 336  Mar 18 1991   dbase
-rw-rw-rw- 1 dupont prof 3999 Oct 21 16:12 demo
drwxrwxrwx 2 dupont prof 704  Nov 13 1990   exoshell
-rw-r--r-- 1 dupont prof 300  Oct 21 16:03  fich
-rw-rw-rw- 1 root  sys  0   Oct 24 16:06  fichier
-rwx------ 1 dupont prof 17 Oct 21 11:09 long
drwxr-xr-x 2 dupont prof 80 Oct 15 14:20 sql
-r--rw-rw- 1 dupont dupont 103 Oct 23 09:23 texte
-rwxrwxrwx 1 dupont projet 59 Oct 24 16:35 a.out
-rw-rw---- 1 dupont projet 59 Oct 24 16:33 test.c
```

```
// Remarque : l'utilisateur dupont fait partie du groupe dupont et d'aucun autre groupe
```

Question 1. L'utilisateur dupont peut-il exécuter `fich` ?

Question 2. L'utilisateur dupont peut-il lire `fichier`, peut-il le modifier ?

Question 3. Un utilisateur du groupe `prof` peut-il lire `fich` ?

Question 4. Un utilisateur du groupe `prof` peut-il modifier `fich` ?

Question 5. Un utilisateur du groupe `prof`, a-t-il le droit de supprimer des fichiers du répertoire `sql` ?

Question 6. L'utilisateur dupont peut-il modifier `texte` ?

Question 7. L'utilisateur dupont peut-il exécuter `long` ?

Question 8. Un utilisateur robert qui n'est pas membre du groupe `prof`, a-t-il le droit de supprimer des fichiers qui ne lui appratiennent pas dans le répertoire `exoshell` ?

Question 9. Un utilisateur du groupe `prof`, a-t-il le droit de renommer des fichiers du répertoire `sql` ?

Question 10. Un utilisateur du groupe `prof`, a-t-il le droit de supprimer le fichier `c` ?

Question 11. Donner la valeur en octale du `umask` permettant de créer des fichier `rw-rw-r-`

Question 12. Donner les droits (littéral et octal) d'un fichier nouvellement créé si la valeur du `umask` est 127.

Question 13. En prenant un umask de 0022, donner les droits du fichier destination pour la copie de fichiers suivante : `$ cp test.c test2.c`

Question 14. En prenant un umask de 0022, donner les droits du fichier destination pour la copie de fichiers suivante : `$ cp test.c a.out`

Vous travaillez avec un collègue appartenant au même groupe que vous.

Question 15. Modifiez les permissions d'un fichier de telle façon que votre collègue puisse le lire et l'exécuter mais ne puisse pas le modifier ni le supprimer.

Question 16. Pouvez-vous modifier les permissions d'un fichier de telle façon que votre collègue puisse le lire, le modifier et l'exécuter alors que vous même ne pouvez le modifier ?

Question 17. Un membre de mon groupe peut-il effacer tout le contenu d'un fichier même si il n'a pas les droits de suppression ?

Question 18. Créer un fichier que votre collègue peut modifier mais pas supprimer et un autre qu'il peut supprimer mais pas modifier.

Question 19. Est-il logique de pouvoir attribuer de tels droits ? Quelles sont les conséquences pratiques de cette expérience ?

Question 20. Combien y a-t-il de combinaisons de droits possibles ?

Travail demandé

Exercice 1 : fichiers et répertoires

Question 21. Donner la commande `ls` (en format long) qui permet d'afficher les répertoires avant les fichiers, ceux-ci étant triés alphabétiquement par leur extension.

Question 22. Donner les formes littérales, numériques puis la valeur du masque pour les politiques de droits d'accès ci-dessous.
– Paranoïaque : seul le propriétaire a le droit de lecture et d'exécution (mais pas de modification)
– Public : Tous les droits pour tous sauf le droit d'écriture pour les autres
– Privé : Seul le propriétaire a tous les droits

Question 23. Expliquer l'intérêt d'une politique paranoïaque.

Question 24. Commenter le danger d'une politique publique qui autoriserait tous les droits pour tous.

Question 25. Commenter l'utilité du *sticky bit* pour cette politique publique.

Question 26. Quelle(s) option(s) de la commande `cp` utiliseriez-vous pour faire des sauvegardes des espaces utilisateurs ?

Question 27. Combien de fichiers de configuration y-a-t-il sur votre système ?

Question 28. Combien de commandes y-a-t-il sur votre système ?

Exercice 2 : surveillance

L'objectif de cet exercice est de s'initier au rôle d'administrateur en réalisant une **tâche de surveillance**.

L'administrateur d'un système n'a pas les mêmes besoins qu'un simple utilisateur. Il est souvent amené à :
– automatiser des traitements longs (par des scripts) et à
– exploiter les services offerts par le système (par les options des commandes)

La commande `find` permet d'effectuer des recherches approfondies dans une arborescence. Elle est souvent utilisée, compte tenu de la richesse de ses critères, en frontal d'une autre commande, pour procéder à la sélection de fichiers. On vous demande d'exploiter les « pouvoirs » de la commande `find` ! Il est donc indispensable de faire un : `$ man find`

En tant qu'administrateur, vous devez surveiller l'ensemble des fichiers des utilisateurs afin de détecter des failles de sécurité potentielles : la première tâche de surveillance est de découvrir les fichiers et répertoires possédant des droits `rwx` pour tous (`o`) ! On limitera la recherche à votre répertoire personnel.

Question 29. Donner la commande `find` exacte.

Question 30. Donner la commande `find` qui permet de rechercher les fichiers qui ne vous appartiennent pas dans votre répertoire personnel.

Question 31. Donner la commande `find` qui permet de rechercher les fichiers qui vous appartiennent dans votre répertoire personnel et dont la taille est supérieure à 10000 octets ou le dernier accès remonte à moins de 30 jours.

Exercice 3 : automatisation d'un traitement

L'objectif de cet exercice est de s'initier au rôle d'administrateur en réalisant une **tâche d'automatisation d'un traitement**.

Il arrive souvent qu'on soit obliger de modifier les droits d'accès d'une arborescence complète en fonction d'une politique qui distingue les fichiers des répertoires. Situations courantes : modification d'une arborescence après une installation d'une application, nouvelle politique pour les espaces utilisateurs, ...

On veut appliquer la politique de droits suivante :
– 640 pour les fichiers contenus dans le répertoire de travail
– 550 pour les répertoires contenus dans le répertoire de travail

Remarque : Si on utilise la commande `chmod` *avec l'option* `-R`*, on ne saura pas distinguer les fichiers des répertoires.*

Là encore, on va avoir recours à la commande `find` !

Question 32. Donner les deux commandes `find` demandées.

