Programmation C TP 1

L'objectif de cette séance est d'aborder les principales commandes UNIX.

Exercice 1 Arborescence et droits

1. Donner la suite de commandes permettant de générer l'arborescence suivante:

```
+tp1/  
  + dir1/  
    -- f11  
    -- f12  
    -- titi  
  + dir2/  
    -- f2
```

2. Mettre les droits suivants:

- tp1: tous les droits pour tout le monde
- dir1: tous les droits pour tout le propriétaire et le groupe
- dir2: accès restreint au propriétaire
- f11: exécutable par tous
- f12: lisible par tous, modifiable par le propriétaire
- f2,titi: lecture et écriture pour le propriétaire

3. Donner la suite de commandes correspondant aux actions suivantes:

- aller dans dir1
- sans changer de répertoire, copier f11 vers tp1
- renommer cette copie en tutu
- aller dans dir2
- déplacer ici tous les fichiers de dir1 commençant par 'f'

Exercice 2 Un peu de compréhension

1. Expliquer ce que fait la suite de commandes suivantes:

```
cd  
mkdir ./tutu  
mkdir tutu/titi  
cd tutu  
chmod u=r ../tutu/titi  
mkdir ~/tutu/titi/toto
```

Exercice 3 Manipulation de fichiers

1. Avec la commande `echo`, créez un fichier nommé `titi` contenant `Bonjour!`
2. En vous aidant de la commande `cat`, créez un fichier `toto` contenant deux fois le contenu de `titi`. Donnez deux façons de faire
3. Que fait la commande `cat z » z ?`
4. Lister l'ensemble des fichiers contenus dans l'arborescence `/var/spool/` en stockant les messages d'erreurs dans un fichier nommé `err`. Affichez les 10 dernières lignes de `err`
5. Donnez des commandes permettant de connaître la taille en octets de la page `man` de `mv` (indice: `ls` donne cette taille)

Exercice 4 Ca va couper chérie ...

1. En utilisant la page `man` de la commande `cut` et en vous rappelant que la sortie de `ls -l` donne quelque chose comme:

```
-rw-r-r- 1 maaouni thesards 289 sep 11 02:48 test.c
```

donnez les commandes utilisant des pipes permettant d'afficher les choses suivantes:
 - juste la liste des droits de ce qu'il y a dans le répertoire (on ne veut pas la première ligne avec "total ...")
 - les droits, propriétaires et noms des fichiers avec l'extension `.c`. Attention, comme `cut` est sensible aux espaces multiples que produit `ls -l`, veillez à les supprimer avec la commande `tr`
 - uniquement les droits en lecture des fichiers
2. Récupérez avec la commande `wget` le fichier suivant:

```
http://igm.univ-mlv.fr/ paumier/C/passwd.txt
```
3. Sachant que les lignes de ce fichier sont de la forme: `talon:x:635:201:Marie-Pierre Talon:/home/compta/talon:/bin/bash`
donnez une commande permettant de sauver la liste des noms et prénoms dans un fichier `tutu`
4. En utilisant `grep`, donnez une commande permettant d'afficher les logins des gens prénommés `Pierre`

Exercice 5 Le tube de la rentrée

1. Lancer en simultané plusieurs instances du programme `xeyes`
2. En utilisant des pipes, n'afficher que les numéros de processus correspondant aux `xeyes`
3. Ajouter un pipe avec la commande `kill` pour tuer ces processus. Que se passe-t-il ?
4. Lorsqu'on met une commande entre quotes inversées (`'toto'`) cette expression est équivalente au résultat affiché par la commande `toto` et va à son tour être évaluée.
Exemple:

```
$>echo find me
find me
$>'echo find me'
find: me: No such file or directory
```

En utilisant des quotes inversées, modifier la commande précédente pour obtenir une ligne de commande qui tue tous les processus xeyes

Exercice 6 Script

1. Un script est un fichier texte contenant des commandes destinées à être interprétées. La première doit contenir un dièse suivi du nom du programme qui va interpréter les commandes. Pour un script shell, cela sera de la forme: `#!/bin/sh`
Créer un script `control.sh` qui affiche "bonjour" quand on le lance
2. En shell, on peut affecter une expression à une variable. Par exemple:

```
cmd='ps'
res='$cmd'
echo -e "$cmd produit le resultat:\n$res"
```

donne comme résultat affiché à l'écran:

```
ps produit le resultat:
  PID TTY          TIME CMD
17607 pts7        00:00:00 bash
28329 pts7        00:00:00 control.sh
28330 pts7        00:00:00 ps
```

Lorsqu'une variable contient plusieurs éléments séparés par des espaces, des tabulations ou des retours à la ligne, on peut parcourir ces éléments au moyen d'une boucle `for` selon le schéma suivant:

```
cmd='ps'
res='$cmd'
for i in $res
do
    echo "*** $i ***"
done
```

Ce qui donne à l'exécution:

```
*** PID ***
*** TTY ***
*** TIME ***
*** CMD ***
*** 17607 ***
*** pts7 ***
```

```

*** 00:00:00 ***
*** bash ***
*** 28536 ***
*** pts7 ***
*** 00:00:00 ***
*** control.sh ***
*** 28537 ***
*** pts7 ***
*** 00:00:00 ***
*** ps ***

```

En utilisant ce qui précède, écrire un script avec une boucle qui affiche tous les fichiers du répertoire courant comme `ls -l`, mais en séparant chaque ligne par des dièses:

```

$>ls -l
total 2
-rwx----- 1 paumier igm 69 sep 5 14:14 control.sh
-rw-r--r-- 1 paumier igm 10915 août 30 11:21 toto
$>./control.sh
#####
-rwx----- 1 paumier igm 69 sep 5 14:14 control.sh
#####
-rw-r--r-- 1 paumier igm 10915 août 30 11:21 toto
#####

```

3. Un script shell peut faire appel aux variables spéciales suivantes:

```

$# nombre de paramètres du script
$0 nom du script
$1 1er paramètre
$2 2eme paramètres
$3 etc
$* liste de tous les paramètres

```

Ecrire un script `param.sh` qui prend deux arguments `a` et `b` qui met dans le fichier `b` le résultat de la commande `a`. Exemple:

```

$>./param.sh ps toto
$>cat toto
  PID TTY          TIME CMD
17607 pts7        00:00:00 bash
28329 pts7        00:00:00 param.sh
28330 pts7        00:00:00 ps

```