Travaux pratiques

1. Gestion des fichiers

But : effectuer des manipulations de base sur le système de fichiers.

1. À partir de votre répertoire personnel créez la structure suivante, en utilisant une seule commande :

```
|-----dossier1
|        |-----dossier3
|-----dossier2
|        |-----dossier4
```

Utilisez la commande **mkdir** avec le paramètre `-p` :

```
$ mkdir -p dossier1/dossier3 dossier2/dossier4
```

2. Déplacez-vous dans le répertoire dossier1 avec un chemin absolu et créez le fichier fichier1 dans ce répertoire.
Le chemin absolu part de la racine sans aucun chemin relatif. Tapez :

```
$ cd /home/user/dossier1
```

Créez le fichier avec touch :

```
$ touch fichier1
```

3. Copiez fichier1 dans le répertoire dossier3 avec un chemin relatif.
Le chemin est relatif en fonction de l'emplacement actuel : .. remonte d'un niveau, et . définit l'emplacement courant.

```
$ pwd
/home/user/dossier1
```

Copiez simplement le fichier dans dossier3, qui est là où vous êtes :

```
$ cp fichier1 dossier3
```

ou encore :

```
$ cp fichier1 ./dossier3
```

4. Déplacez-vous dans dossier2 en utilisant un chemin relatif, et copiez le fichier fichier1 de dossier3 sous le nom fichier2 là où vous êtes.
Déplacez-vous :

```
$ cd ../dossier2
```

Copiez le fichier :

```
$ cp ../dossier1/dossier3 ./fichier2
```

5. Renommez et déplacez fichier2 en fichier3 dans le répertoire dossier3.
La commande **mv** déplace et renomme.

```
$ mv fichier2 ../dossier1/dossier3/fichier3
```

6. Supprimez fichier1 du répertoire dossier3.
   La commande **rm** supprime le fichier.

```
$ rm ../dossier1/dossier3/fichier1
```

7. Avec rmdir supprimez dossier2, puis dossier1 et tout son contenu. Est-ce possible ? Pourquoi ? Comment faire ?
   Vous ne pouvez pas supprimer dossier2 directement avec rmdir car il contient dossier4 et n'est donc pas vide. Vous devez donc passer par la commande **rm** avec le paramètre `-r`.

```
$ cd
$ rm -rf dossier2
```

8. Quel est le but de la commande `ls -l [a-z]*. ??[ !0-9]` ?
   Le fichier commence par une lettre de a à z et finit par quatre caractères : le point, deux caractères quelconques et le dernier étant n'importe quoi sauf un chiffre. Les détails des fichiers qui correspondent à ce critère sont affichés.

9. Créez un fichier appelé « -i » avec une redirection : echo > -i. Tentez de le supprimer.
   C'est un classique. Si vous tentez de supprimer le fichier, rm retourne une erreur indiquant qu'il manque un paramètre (le nom du fichier). Il interprète -i comme étant une option de la commande. La question est donc : comment faire en sorte que le -i ne soit pas reconnu comme une option ?
   Étant donné que tout ce qui commence par un tiret est considéré comme un paramètre, indiquez un chemin permettant d'isoler le tiret :

```
$ rm ./-i
```

2. Rechercher des fichiers

But : rechercher des fichiers avec find, whereis et locate.

1. Affichez tous les fichiers ayant une taille inférieure à 400 octets et ayant les droits 644.
   Utilisez les paramètres `-size` et `-perm` de la commande **find** :

```
$ find / -size -400c -perm 644 -print
```

2. Affichez tous les fichiers dans votre répertoire personnel ayant une taille inférieure à 400 blocs.

```
$ find ~ -size -400 -print
```

3. Listez en format long tous les fichiers du système vous appartenant modifiés il a plus de 7 jours.
   Utilisez les paramètres `-user` et `-mtime` :

```
$ find / -user seb -mtime +7 -ls
```

4. Listez et affichez en format long les fichiers dans votre répertoire personnel qui ont comme propriétaire guest ou qui ont une taille entre 512 et 1024 octets, inclus.
   Le petit piège réside ici dans le "inclus". Si vous indiquez +512c, les fichiers de 512 octets sont exclus. Vous devez modifier les bornes en conséquence.

```
$ find ~ -user guest -size +511c -size -1025c -ls
```

5. Recherchez tous les fichiers vides du système n'appartenant pas à root et tentez de les supprimer.
   Utilisez les paramètres `-empty` et `-exec` pour exécuter un rm sur chaque fichier trouvé.

```
$ find / -type f -empty -exec rm -f {} \;
```

3. Les redirections

But : manipuler les redirections de canaux.

1. La commande **find /** retourne beaucoup d'erreurs si elle est utilisée par un simple utilisateur à cause d'un problème de droits. Évitez les messages d'erreurs en les redirigeant vers un « trou noir » :

```
$ find / 2>/dev/null
```

2. Dans le cas précédent et malgré les erreurs, vous avez encore accès à beaucoup d'emplacements et la liste qui s'affiche est très longue et donc inexploitable. Placez cette liste dans un fichier appelé résultat.

```
$ find / 1>resultat 2>/dev/null
```

3. Maintenant, plus rien ne s'affiche. En fin de compte, pour savoir pourquoi vous ne pouvez pas accéder à certains répertoires vous voulez aussi obtenir les messages d'erreurs dans le fichier résultat, avec la liste des fichiers. Faites une redirection du canal d'erreur standard dans le canal de sortie standard :

```
$ find / >resultat 2>&1
```

4. Plus rien ne s'affiche. Vous voulez les deux : un fichier et l'affichage des résultats sur écran. La commande **tee** s'utilise avec un tube et permet de récupérer un flux sortant, de le placer dans un fichier et de ressortir ce flux comme si de rien n'était :

```
$ find / 2>&1 | tee resultat.
```

Votre fichier résultat contient la liste de tous les fichiers accessibles, les erreurs et le tout s'affiche aussi sur l'écran.

4. Les filtres et utilitaires

But : manipuler les filtres et utilitaires sur un fichier classique.

1. Le fichier /etc/passwd est un grand classique sous Unix. Il se compose de sept champs séparés par des « : » : `login:passwd:UID:GID:Commentaire:homedir:shell`. Récupérez la ligne de l'utilisateur root dans /etc/passwd :

```
$ grep ^root: /etc/passwd
```

2. De cette ligne, récupérez l'UID de root :

```
$ grep ^root: /etc/passwd | cut -d: -f3
```

3. Comptez le nombre d'utilisateurs contenus dans ce fichier à l'aide d'une redirection en entrée :

```
$ wc -l < /etc/passwd
```

4. Un peu plus compliqué : récupérez la liste des GID, triez-les par ordre croissant et supprimez les doublons :

```
$ cut -d: -f4 /etc/passwd | sort -n | uniq
```

5. De là, extrapolez le nombre de groupes différents utilisés :

```
$ cut -d: -f4 /etc/passwd | sort -n | uniq | wc -l
```

6. Convertissez tous les logins en majuscules :

```
$ cut -d: -f1 /etc/passwd | tr "[a-z]" "[A-Z]"
```

7. Isolez maintenant la huitième ligne de /etc/passwd. Il y a plusieurs solutions, en voici deux :
`$ head -8 /etc/passwd | tail -1`
Et :
`$ grep -n "" /etc/passwd | grep ^8: | cut -d: -f2-`

5. Les processus

But : gérer les processus.

1. Lancez le processus sleep 1000 en arrière-plan. Récupérez son PID.

```
$ sleep 1000&
[1] 9168
```

(le PID varie bien entendu).

2. Replacez ce processus en avant-plan, puis stoppez-le (ne le tuez pas) et replacez-le en arrière-plan.

```
$ fg
sleep 1000
(CTRL-Z)
[1]+  Stopped                 sleep 1000
$ bg
[1]+ sleep 1000 &
```

3. Indiquez les détails de ce processus :

```
$ ps p 9168 -f
UID        PID  PPID  C STIME TTY      STAT   TIME CMD
seb       9168  8096  0 10:46 pts/1    S      0:00 sleep 1000
```

4. Modifiez la priorité de ce processus ; passez-la à un facteur 10 :

```
$ renice 10 9168
9168: priorité précédente 0, nouvelle priorité 10
```

5. Listez à nouveau le détail de ce processus mais au format long. Regardez la valeur de la colonne NI :

```
$  ps p 9168 -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY      TIME CMD
0 S  1000  9168  8096  0  90  10 -  2324 restar pts/1    0:00
sleep 1000
```

6. Envoyez le signal 15 à ce processus. Ceci va le terminer.

```
$ kill -15 9168
[1]+  Complété              sleep 1000
```

6. Programmation Shell Niveau 1

But : dans un fichier texte nous avons les lignes suivantes :

1 3

5 7

12 19

...

Écrivez un script qui accepte ce fichier comme paramètre, qui le lit et pour chacune de ses lignes calcule la somme des deux nombres et l'affiche sous la forme suivante :

1 + 3 = 4

5 + 7 = 12

12 + 19 = 31

...

1. Vérifiez en début de script que le nombre de paramètres passé au script est égal à 1 et que ce paramètre est bien un fichier.

```
[ $# -ne 1 -o ! -f $1 ] && exit
```

2. Initialisez une variable à 0 qui contiendra le total de chacune des lignes.

```
result=0
```

3. Le fichier doit être lu ligne à ligne ; écrivez une boucle qui lit une ligne tant que la fin du fichier n'est pas atteinte :

```
while read ligne
do
...
done < $1
```

4. Dans la boucle, récupérez les deux valeurs des lignes, le séparateur est l'espace. Placez les deux valeurs dans les variables c1 et c2 :

```
while read ligne
do
     c1=$(echo $ligne | cut -d" " -f1)
     c2=$(echo $ligne | cut -d" " -f2)
     ...
done < $1
```

5. Additionnez ces deux valeurs et placez le résultat dans result. Affichez result.

```
[ $# -ne 1 -o ! -f $1 ] && exit
result=0
while read ligne
do
     c1=$(echo $ligne | cut -d" " -f1)
     c2=$(echo $ligne | cut -d" " -f2)
     result=$((c1+c2))
     echo -e "$c1 + $c2 = $result"
done < $1
```

7. Fonction Shell

But : un nombre est premier seulement si ses seuls diviseurs sont 1 et lui-même.

Autrement dit, si on peut diviser un nombre par autre chose que par un et lui-même et que le résultat de cette division est un entier (ou que le reste de cette division est 0, ce qui revient au même) alors il n'est pas premier.

Le chiffre 1 n'est pas premier car il n'a pas deux diviseurs différents.

Le chiffre 2 est premier (2 x 1 donc deux diviseurs différents).

Aucun nombre pair (sauf le chiffre 2) n'est premier (car ils sont tous divisibles par 2).

Liste des dix premiers nombres premiers : 2 3 5 7 11 13 17 19 23 29.

Écrivez une fonction appelée **est_premier**, qui prend comme paramètre un nombre et qui renvoie le code retour 0 si le nombre est premier, 1 sinon.

```
est_premier()
{
        [ $1 -lt 2 ] && return 1
        [ $1 -eq 2 ] && return 0
        [ $(expr $1 % 2) -eq 0 ] && return 1
        i=3
        while [ $((i*i)) -le $1 ]
        do
                [ $(expr $1 % $i) -eq 0 ] && return 1
                i=$((i+2))
        done
        return 0
}
```