TP 1 – Les bases du Shell

Pour ces exercices, n'oubliez pas que vous pouvez utiliser :

la complétion touche tabulation (tab)

- 1 `tab` $\Longrightarrow$ complétion si pas de conflit,
- 2 `tab` en cas de conflit $\Longrightarrow$ affichage des possibilités de complétion.

le rappel de commandes touches flèches : $\uparrow$ et $\downarrow$.

la modification de commandes touches flèches : $\leftarrow$ et $\rightarrow$.

Dans ce document, <ENTER> signifie la pression de la touche entrée du clavier, et <CTRL-X> signifie la pression simultanée des touches du clavier contrôle (ctrl) et x.

Les commandes que vous devez taper seront données sous la forme

```
sh$ atril /pub/FISE_OSSE11/Supports/tp1-suj.pdf &
```

La partie "**sh$** " ci-dessus représente votre prompt Shell (le message déjà affiché à l'écran), il ne faut donc pas la taper.

Vous devez faire au moins les exercices 1 à 13.
Les exercices 14 et 15 sont issus des devoirs sur table donnés les années précédentes.
L'exercice 16 propose des questions un peu plus difficiles pour celles et ceux qui avancent vite.
L'exercice 17 est un complément.

Exercice 1 - `cd`, `pwd`, HOME, `xterm`, `exit`

1. Dans un terminal, créez et placez vous dans le répertoire ~/sys/tp1.

   ```
   sh$ mkdir -pv ~/sys/tp1
   ```

   ```
   sh$ cd ~/sys/tp1
   ```

2. Affichez le répertoire courant (*Working Directory*).

   ```
   sh$ pwd
   ```

3. Allez dans votre répertoire personnel (HOME).

   ```
   sh$ cd
   ```

4. Affichez le répertoire courant.

   ```
   sh$ pwd
   ```

5. Retournez dans le répertoire ~/sys/tp1.

   ```
   sh$ cd -
   ```

6. Exécutez la commande `ls` sans argument dans votre HOME, sans changez de répertoire courant, de quatre manières différentes.

   ```
   sh$ ( cd ; ls )
   ```

   ```
   sh$ ( cd ~ ; ls )
   ```

   ```
   sh$ ( cd $HOME ; ls )
   ```

   ```
   sh$ ( cd ../.. ; ls )
   ```

7. Ouvrez un nouveau terminal en tâche de fond.

   ```
   sh$ xterm &
   ```

8. Vérifiez que les terminaux ont le même répertoire courant.

```
sh$ pwd # dans le premier terminal
```

```
sh$ pwd # dans le deuxième terminal
```

9. Fermez le deuxième terminal sans cliquer avec la souris.

```
sh$ exit # dans le deuxième terminal
```

10. Ouvrez à nouveau un deuxième terminal.

```
sh$ xterm
```

11. Que se passe-t-il au niveau du premier terminal si on ne met pas le & ?

12. Fermez le deuxième terminal à l'aide d'un raccourci clavier.

```
en tapant <CTRL-D> dans le deuxième terminal
```

Exercice 2 - `cp`, `mv`, `mkdir`, `ls`

1. Ouvrez un terminal et placez vous dans votre HOME.

```
sh$ cd
```

2. Sans changer de répertoire courant, copiez le fichier `/pub/FISE_OSSE11/shell/fich.dat` dans le répertoire `/tmp` sous le nom `f1`.

```
sh$ cp -v /pub/FISE_OSSE11/shell/fich.dat /tmp/f1
```

3. Quelle est la taille en octet de `/tmp/f1` ?

```
sh$ ls -l /tmp/f1
```

4. Sans changer de répertoire courant, déplacez le fichier `/tmp/f1` dans `/tmp/d1/d2/f1-de-d2` (il faudra créer le répertoire /tmp/d1/d2).

```
sh$ mkdir -pv /tmp/d1/d2
```

```
sh$ mv -v /tmp/f1 /tmp/d1/d2/f1-de-d2
```

5. Sans changer de répertoire courant, copiez l'arborescence `/tmp/d1/d2` dans votre HOME sous le nom `sys/tp1/exo2`.

```
sh$ mkdir -pv sys/tp1
```

```
sh$ cp -rv /tmp/d1/d2 sys/tp1/exo2
```

6. Sans changer de répertoire courant, déplacez l'arborescence `/tmp/d1` dans le répertoire `sys/tp1/exo2` de votre HOME

```
sh$ mv -v /tmp/d1 sys/tp1/exo2
```

7. Vérifiez que le répertoire `sys/tp1/exo2` contient le fichier `f1-de-d2` et le répertoire `d1`.

```
sh$ ls -l sys/tp1/exo2
```

8. Vérifiez, en une seule commande, que les deux fichiers `sys/tp1/exo2/f1-de-d2` et `sys/tp1/exo2/d1/d2/f1-de-d2` sont bien de même taille.

```
sh$ ls -l sys/tp1/exo2{,/d1/d2}/f1-de-d2
```

Exercice 3 - `cat`, `echo`, sortie redirigée vers un fichier

Le but de cet exercice est de créer un fichier `a` contenant 1000 fois la lettre 'A'.

1. Placez-vous dans le répertoire `/tmp`.

```
sh$ cd /tmp
```

2. Créez le fichier `a` qui contient une seule lettre 'A'.

```
sh$ echo -n 'A' > a
```

```
sh$ cat a
sh$ ls -l a
```

3. Créez un nouveau fichier `a` qui contient 10 fois le contenu actuel du fichier `a` (on utilisera un fichier intermédiaire `/tmp/1`).

```
sh$ cat a a a a a a a a a a > /tmp/1 ; mv /tmp/1 a
```

4. Répétez la commande précédente une seconde fois.

```
Utilisez votre historique de commandes.
```

5. Vérifiez le contenu et la taille du fichier `a`.

```
sh$ cat a
sh$ ls -l a
```

6. Faites en sorte d'avoir 1000 fois 'A' dans `/tmp/1`.

Exercice 4 - Variable d'environnement $HOME

1. Depuis un terminal dont le répertoire courant est `/tmp`, créez un nouveau répertoire `~/sys/exo4` dans votre HOME.

```
sh$ cd /tmp
```

```
sh$ mkdir -pv ~/sys/exo4
```

2. Ouvrez un second terminal dont le HOME est ce répertoire `exo4`.

```
sh$ HOME=~/sys/exo4 xterm &
```

3. Dans ce second terminal, allez dans votre HOME et vérifiez qu'il s'agit bien du répertoire `exo4`.

```
sh$ cd
```

```
sh$ pwd ; echo ~ $HOME
```

4. De ce second terminal, allez dans `/tmp`, et créez le fichier `f1` vide dans votre vrai HOME.

```
sh$ cd /tmp
sh$ touch ~/../../f1
```

5. Du premier terminal, allez dans `/tmp`, et vérifiez qu'il existe bien un fichier vide `f1` dans votre HOME, puis supprimez le.

```
sh$ cd /tmp
sh$ ls -l ~/f1
sh$ rm -v ~/f1
```

6. Fermez le second terminal sans utiliser la souris.

Exercice 5 - Variable d'environnement $PATH

1. Créez dans votre HOME le répertoire `bin`. Copiez dans ce répertoire les commandes `ls` et `cat`, sous les noms `myls` et `mycat`.

```
sh$ mkdir -pv ~/bin
sh$ cp -v /bin/ls ~/bin/myls
sh$ cp -v /bin/cat ~/bin/mycat
```

2. Lancez les commandes `myls`, `mycat`, `~/bin/myls` et `~/bin/mycat`.

3. Ajoutez ce nouveau répertoire `bin` à votre PATH.

```
sh$ export PATH="$PATH:~/bin"
```

4. Lancez les commandes `myls` et `mycat` (cela doit fonctionner cette fois).

5. Créez un nouveau terminal.

```
sh$ /usr/bin/xterm &
```

   Dans ce terminal, les commandes `myls` et `mycat` fonctionnent elles encore ? Expliquez pourquoi.

6. Depuis le bureau, ouvrez un nouveau `xterm` à l'aide du raccourci clavier <ALT-F2>. Dans ce terminal, les commandes `myls` et `mycat` fonctionnent elles encore ? Expliquez pourquoi.

Exercice 6 - `less`, `cat`, `echo`, calcul, sous-shell

1. Ouvrez un terminal et placez vous dans votre HOME.

```
sh$ cd
```

2. Visualisez le contenu du fichier `/pub/FISE_OSSE11/shell/expr-arith.dat`.

```
sh$ cat /pub/FISE_OSSE11/shell/expr-arith.dat
```

```
sh$ less /pub/FISE_OSSE11/shell/expr-arith.dat # q pour quitter
```

3. Calculez cette expression.

```
sh$ echo $(( $(cat /pub/FISE_OSSE11/shell/expr-arith.dat) ))
```

4. Créez le fichier `/tmp/expr1` contenant cette expression multipliée par elle même en utilisant la commande `echo`.

```
sh$ echo "( $(cat /pub/FISE_OSSE11/shell/expr-arith.dat) )
                * ( $(cat /pub/FISE_OSSE11/shell/expr-arith.dat) )" > /tmp/expr1
```

5. Calculez l'expression de `/tmp/expr1`.

```
sh$ echo $(( $(cat /tmp/expr1) ))
```

6. Créez le fichier `/tmp/expr2` contenant l'expression de `/pub/FISE_OSSE11/shell/expr-arith.dat` multipliée 3 fois par elle même en utilisant la commande `echo` et une variable.

```
sh$ x="($(cat /pub/FISE_OSSE11/shell/expr-arith.dat))"
```

```
sh$ echo $x \* $x \* $x > /tmp/expr2
```

7. Calculez l'expression de `/tmp/expr2`.

```
sh$ echo $(( $(cat /tmp/expr2) ))
```

8. Créez le fichier `/tmp/expr3` contenant l'expression qui est la somme des expressions de `/tmp/expr1` et `/tmp/expr2` en utilisant la commande `cat`.

```
sh$ cat - /tmp/expr1 - /tmp/expr2 - >/tmp/expr3
(<ENTER>
<CTRL-D>)+(<ENTER>
<CTRL-D>)<ENTER>
<CTRL-D>sh$
```

note : L'argument '-' est interprété par la commande cat comme le flux stdin.

9. Calculez l'expression de `/tmp/expr3`

10. Supprimez les fichiers `expr*` du répertoire `/tmp`.

```
sh$ rm -v /tmp/expr*
```

Exercice 7 - `fg`, `bg`, processus en avant-plan et en arrière-plan

Le but de cet exercice est d'illustrer le concept de processus en avant-plan et en arrière-plan.
Le raccourci clavier <CTRL-C> envoie une demande de terminaison au processus en avant-plan. Nous allons commencer par voir l'effet de ce raccourci dans différentes situations.

1. Lancez un `atril` en avant-plan (foreground),

   ```
   sh$ atril /pub/FISE_OSSE11/Supports/sys-poly.pdf
   ```

   puis tapez <CTRL-C> dans le terminal.

   Le `atril` reçoit la demande et se ferme.

2. Tapez à nouveau <CTRL-C> dans le terminal.

   Il ne se passe rien. Ici, le programme en avant-plan est `bash`, et ce dernier ignore les demandes de terminaison reçues.

3. Lancez un `atril` en arrière-plan (background), puis tapez <CTRL-C> dans le terminal.

   ```
   sh$ atril /pub/FISE_OSSE11/Supports/sys-poly.pdf &
   ```

   Le `atril` ne se ferme pas. En effet, le programme en avant-plan est `bash`, donc `atril` n'a pas reçu de demande de terminaison. D'ailleurs, vous pouvez exécuter des commandes dans le terminal, par exemple `ls`, tout en utilisant `atril`.

4. Ramenez le `atril` en avant-plan (foreground) grâce à la commande builtin `fg`, qui ramène en avant-plan le dernier processus lancé en arrière-plan.

   ```
   sh$ fg
   ```

   Désormais, `atril` est en avant-plan et pourrait donc être fermé grâce à <CTRL-C>. Dans le terminal, `bash` n'est plus en avant-plan.

5. Tapez `ls` puis <ENTER> dans le terminal, et quittez `atril` via son menu. Expliquez ce qui se passe dans le terminal.

6. Lancez `xterm` en avant-plan.

   ```
   sh$ xterm
   ```

7. Ceci n'était pas une très bonne idée (pourquoi ?). Nous allons donc ramener `xterm` en arrière-plan. Ceci ce fait en 2 étapes :

   i Tapez <CTRL-Z> (ce qui envoie la demande au processus en avant-plan de se mettre en pause). `xterm` reçoit et obéit à la demande de mise en pause. Du coup, `bash` revient en avant-plan et il est possible de taper des commandes.

   ii Tapez la commande

   ```
   sh$ bg
   ```

   La commande builtin `bg` réveille et fait passer en arrière plan le dernier processus mis en pause (ici `xterm`). `bash` reste en avant-plan.

Exercice 8 - `cut`, `head`, `tail`, `sed`

Le but de cet exercice est d'introduire, à l'aide d'exemples, plusieurs commandes très utiles pour manipuler des fichiers.

Ces commandes prennent toutes un certain nombre d'options puis un fichier en argument. Le résultat est renvoyé sur la sortie standard, et donc par défaut affiché à l'écran. Si aucun fichier n'est précisé, ces commandes travaillent sur les données issues de l'entrée standard. Ainsi, il est possible de chaîner plusieurs traitements en utilisant plusieurs de ces commandes, séparées par des |.

1. Affichez le contenu du fichier `/pub/FISE_OSSE11/shell/select.dat` (commande `less` ou `cat`).
   Les blancs que vous voyez sont formés d'un seul caractère de tabulation.

2. Sélections de colonnes avec la commande `cut` :
   a. Affichez la 3e colonne du fichier `/pub/FISE_OSSE11/shell/select.dat`.
   ```
   sh$ cut -f 3 /pub/FISE_OSSE11/shell/select.dat
   ```
   b. Affichez la première et la troisième colonne de `/pub/FISE_OSSE11/shell/select.dat`.
   ```
   sh$ cut -f 1,3 /pub/FISE_OSSE11/shell/select.dat
   ```
   c. Qu'obtient on si on essaie d'échanger les colonnes 1 et 3 en écrivant `3,1` au lieu de `1,3` dans la commande précédente ?
   d. Affichez la liste des numéros de groupes (GID). Cette information est disponible dans la quatrième colonne du fichier `/etc/passwd`.

3. Sélections de lignes avec les commandes `head`, `tail` et `grep` :
   a. Affichez les deux premières lignes de `/pub/FISE_OSSE11/shell/select.dat`.
   ```
   sh$ head -n 2 /pub/FISE_OSSE11/shell/select.dat
   ```
   b. Affichez les trois dernières lignes de `/pub/FISE_OSSE11/shell/select.dat`.
   ```
   sh$ tail -n 3 /pub/FISE_OSSE11/shell/select.dat
   ```
   c. Affichez les lignes du fichier `/pub/FISE_OSSE11/shell/select.dat` contenant le mot `Alice`.
   ```
   sh$ grep -e Alice /pub/FISE_OSSE11/shell/select.dat
   ```

4. Manipulations de fichiers avec la commande `sed` :
   a. Affichez les trois premières lignes de `/pub/FISE_OSSE11/shell/select.dat`.
   ```
   sh$ sed -e '4,$d' /pub/FISE_OSSE11/shell/select.dat
   ```
   Ici, `d` signifie *delete*, et cette action s'applique de la ligne 4 (incluse) à la fin du fichier (représenté dans la commande par le symbole $).
   b. Affichez le fichier `/pub/FISE_OSSE11/shell/select.dat` privé de ses trois premières lignes.
   ```
   sh$ sed -e '1,3d' /pub/FISE_OSSE11/shell/select.dat
   ```
   c. Affichez `/pub/FISE_OSSE11/shell/select.dat` privé de ses deux dernières lignes. Pour cela, il faut au préalable calculer le nombre de lignes du fichier.
   ```
   sh$ nb=$(wc -l </pub/FISE_OSSE11/shell/select.dat)
   sh$ sed -e "$((nb-1)),\$d" /pub/FISE_OSSE11/shell/select.dat
   ```
   d. Que se passe-t-il si on oublie de mettre un \ devant le symbole $?
   e. Affichez les lignes de `/pub/FISE_OSSE11/shell/select.dat` ne contenant pas le mot `Alice`.
   ```
   sh$ sed -e '/Alice/d' /pub/FISE_OSSE11/shell/select.dat
   ```
   Ici, la partie `/Alice/` permet de sélectionner les lignes contenant le mot `Alice`, auxquelles on applique l'action de suppression `d`.
   f. Affichez les lignes de `/pub/FISE_OSSE11/shell/select.dat` contenant le mot `Alice`.
   ```
   sh$ sed -e '/Alice/!d' /pub/FISE_OSSE11/shell/select.dat
   ```

Le symbole `!` permet ici d'inverser la sélection, et donc d'appliquer l'action `d` à toutes les lignes ne contenant pas le mot `Alice`.

Une solution alternative pour la question précédente est d'utiliser la commande

```
sh$ sed -n -e /Alice/p /pub/FISE_OSSE11/shell/select.dat
```

où l'action `p` signifie `print`, et où on a utilisé l'option `-n` de `sed` pour ne pas afficher les lignes qui n'ont pas été sélectionnées (cf man).

g. Affichez le contenu du fichier `/pub/FISE_OSSE11/shell/select.dat`, mais en remplaçant `Alice` par `Anne`.

```
sh$ sed -e 's/Alice/Anne/g' /pub/FISE_OSSE11/shell/select.dat
```

Ici, l'action de `sed` est `s` pour *substitute*. Sa syntaxe est

`s/motif1/motif2/options`

et elle a pour but de remplacer le motif `motif1` par `motif2`.

Par défaut, seule la première occurrence de `motif1` de chaque ligne est remplacée. Nous avons donc utilisé l'option `g` (*global*) pour remplacer **toutes** les occurrences de chaque ligne.

h. Affichez le contenu du fichier `/pub/FISE_OSSE11/shell/select.dat`, mais en remplaçant la date du `01/08/2001` par `03/08/2001`.

```
sh$ sed -e 's,01/08/2001,03/08/2001,' /pub/FISE_OSSE11/shell/select.dat
```

Comme nos motifs contiennent le symbole /, nous avons utilisé le symbole , comme séparateur entre les différents éléments de syntaxe pour `s`. On peut en fait utiliser le caractère que l'on veut comme séparateur, les choix les plus courants étant /, , et |.

i. Affichez la première colonne de `/pub/FISE_OSSE11/shell/select.dat`.

```
sh$ sed -e 's/\t.*//' /pub/FISE_OSSE11/shell/select.dat
```

Les motifs peuvent contenir des caractères spéciaux. Ici, `\t` désigne le symbole de tabulation, le `.` est un joker pour désigner n'importe quel caractère (sauf un saut de ligne), et `*` signifie de répéter le motif associé (ici `.`) un nombre quelconque (potentiellement nul) de fois. Ainsi, `.*` nous permet de désigner une suite quelconque de caractères.

On notera que `sed` va toujours considérer le motif le plus grand possible. Ici, le `\t` considéré sera le premier d'une ligne, de sorte que le motif complet couvrira les deux dernières colonnes.

j. Affichez la troisième colonne de `/pub/FISE_OSSE11/shell/select.dat`.

```
sh$ sed -e 's/[^\t]*\t[^\t]*\t\([^\t]*\)/\1/'
                                        /pub/FISE_OSSE11/shell/select.dat
```

Ici, `[^\t]` correspond à tout caractère sauf le symbole de tabulation (et un saut de ligne). Le contenu d'un colonne est donc désigné par `[^\t]*`.

L'utilisation de \(et \) permet d'affecter un indice i au contenu entre parenthèses. Il est possible d'utiliser ce contenu pour le motif cible via $\backslash i$.

k. Affichez la première et la troisième colonne du fichier `/pub/FISE_OSSE11/shell/select.dat`.

```
sh$ sed -e 's/\([^\t]*\)\t[^\t]*\t\([^\t]*\)/\1\t\2/'
                                        /pub/FISE_OSSE11/shell/select.dat
```

l. Échangez la première et la troisième colonne du fichier.

```
sh$ sed -e 's/\([^\t]*\)\t[^\t]*\t\([^\t]*\)/\2\t\1/'
                                        /pub/FISE_OSSE11/shell/select.dat
```

m. La commande `sed` est un outil très utile en pratique. Les personnes curieuses pourront consulter le site `https://www.grymoire.com/Unix/Sed.html` , qui présente de nombreux autres usages de cette commande.

5. À vous de jouer :

a. Donnez la première ligne du fichier `/pub/FISE_OSSE11/shell/select.dat` qui contient le mot `Alice` en utilisant `grep` et `head`.

b. Donnez la première ligne du fichier `/pub/FISE_OSSE11/shell/select.dat` qui contient le mot `Alice` en utilisant `sed` et éventuellement `head`.

   note : L'action `q` (pour *quit*) de `sed` permet d'arrêter le traitement en cours.

c. Donnez la destination de la première ligne qui contient `Alice` en utilisant `grep`, `head` et `cut`.

d. Donnez la destination de la première ligne qui contient `Alice` en utilisant `sed` et `head`.

e. Donnez la destination de la dernière ligne qui contient `Alice`.

Exercice 9 - `wc`, `stat`

L'objectif de cet exercice est de mettre dans la variable `sz` la taille du fichier `/pub/FISE_OSSE11/shell/gen-projet.sh`. Nous proposons ici deux approches différentes.

1. Utilisation de l'option `-c` de la commande `wc` :
   a. Donnez la taille en octet de `/pub/FISE_OSSE11/shell/gen-projet.sh`.
   ```
   sh$ wc -c /pub/FISE_OSSE11/shell/gen-projet.sh
   ```
   La commande écrit le nom du fichier, ce qui oblige à le supprimer avec une autre commande (`sed` ou `cut`).
   b. Donnez uniquement la taille en octet du fichier `/pub/FISE_OSSE11/shell/gen-projet.sh`.
   ```
   sh$ wc -c < /pub/FISE_OSSE11/shell/gen-projet.sh
   ```
   c. Définissez la variable `sz` avec la bonne valeur.
   ```
   sh$ sz=$(wc -c < /pub/FISE_OSSE11/shell/gen-projet.sh)
   sh$ echo "sz = $sz"
   ```
   Cette approche a un inconvénient majeur. La commande `wc` va lire et compter les caractères du fichier un par un. Sur un fichier de grande taille, le surcoût engendré n'est tout simplement pas acceptable.

2. Utilisation de la commande `stat` :
   a. Une approche plus efficace est de récupérer directement les informations relative à notre fichier à l'aide de la commande `stat`. On peut préciser le format de sortie grâce à l'option `-c <format>`. Comme seule la taille nous intéresse ici, on va utiliser `%s` comme format.
   ```
   sh$ sz=$(stat -c '%s' /pub/FISE_OSSE11/shell/gen-projet.sh)
   sh$ echo "sz = $sz"
   ```
   b. Consultez le man de `stat` et inspirez vous de ce qui précède pour obtenir la taille réellement utilisée sur le disque.

   aide :
   - Ouvrir le man avec `sh$ man stat`
   - Tapez `/size`<ENTER> (la touche `/` étant le raccourci pour passer en mode recherche).
   - La touche `n` permet d'aller à la prochaine occurrence du mot `size`, jusqu'à arriver à la dernière occurrence du fichier.
   - La touche `N` permet d'aller à l'occurrence précédente du mot `size`.
   - Un fichier est stocké sur le disque en différents morceaux (de même taille) appelés blocs.
   - La séquence `'%S'` n'est valable que pour les *systèmes de fichiers*, comme on peut le voir en remontant un peu dans la documentation.
     Comme `/pub/FISE_OSSE11/shell/gen-projet.sh` est un fichier (et sûrement pas un *système de fichiers*), cette séquence ne fonctionnera pas ici.
   - La syntaxe `$(( ))` va être utile ici.

Exercice 10 - `sort`, `uniq`, `wc`

1. Affichez le contenu du fichier `/pub/FISE_OSSE11/shell/unsort.dat` (commande `less` ou `cat`).

2. Triez le fichier `/pub/FISE_OSSE11/shell/unsort.dat` à l'aide de la commande `sort` et stockez le résultat dans le fichier `/tmp/1`.

3. Vérifiez que le fichier `/tmp/1` a autant de lignes que le fichier `/pub/FISE_OSSE11/shell/unsort.dat`.

```
sh$ wc -l /pub/FISE_OSSE11/shell/unsort.dat
sh$ wc -l </tmp/1
```

   Expliquez la différence entre les deux syntaxes proposées.

4. Générez `/tmp/2` à partir `/tmp/1` en enlevant les lignes semblables grâce à la commande `uniq`.

5. Consultez le man de `sort` pour trouver comment obtenir le résultat de `/tmp/2` depuis `/pub/FISE_OSSE11/shell/unsort.dat` en utilisant une seule commande.

6. Calculez le nombre de lignes enlevées grâce à la suppression des lignes en doublons.

   note : On pourra stocker les tailles des différents fichiers dans des variables et effectuer plusieurs commandes.

7. Refaites le même calcul sans créer de fichier intermédiaire et en une seule commande (utilisez des pipes).

Exercice 11 - `find`

La commande `find` permet de sélectionner des fichiers en combinant certains critères et d'exécuter une action sur chacun des fichiers sélectionnés.

La syntaxe générale est

`find <repertoire> <filtre1> ... <filtre`n`> <action1> ... <action`k`>`

où le premier argument est le répertoire dans lequel on veut faire la recherche. On peut omettre de préciser l'action (cas $k = 0$). Dans ce cas, `find` se contente d'afficher les noms des fichiers sélectionnés (ce qui correspond à l'action `-print`).

1. Affichez tout ce qui est contenu dans votre répertoire `HOME`.

   ```
   sh$ find $HOME -print
   ```

   ou

   ```
   sh$ ( cd ; find . -print )
   ```

   La seconde forme, qui donne des chemins relatifs, est en général plus utile.

2. Affichez tous les fichiers (réguliers) du répertoire courant.

   ```
   sh$ find . -type f -print
   ```

3. Affichez tous les répertoires du répertoire courant.

   ```
   sh$ find . -type d -print
   ```

4. Affichez tous les fichiers (réguliers) du répertoire courant dont la taille est supérieure à 10ko.

   ```
   sh$ find . -type f -size +10k -print
   ```

5. Affichez tous les fichiers (réguliers) du répertoire courant dont la taille est inférieure à 100 octets.

   ```
   sh$ find . -type f -size -100c -print
   ```

6. Affichez tous les fichiers (réguliers) du répertoire courant dont les noms se terminent par `.c` ou `.h`.

   ```
   sh$ find . -type f -name '*.[hc]' -print
   ```

   Le paramètre de l'option `-name` est une expression régulière qui porte sur le nom de base du fichier. On peut aussi utiliser `-iname` pour faire une recherche insensible à la casse (minuscule ou majuscule).

7. Pourquoi doit-on mettre des ' dans la commande précédente ?

8. Recherchez, parmi les fichiers `.c` du répertoire `/pub/FISE_OSSE11/shell/projet`, ceux qui contiennent un appel à la fonction `strcmp`.

   ```
   sh$ find /pub/FISE_OSSE11/shell/projet -type f -name '*.c'
                                                -exec grep -w -l -e strcmp {} \;
   ```

   L'option `-exec` (très utilisée en pratique) permet d'exécuter une commande sur chaque fichier sélectionné, en remplaçant d'abord `{}` par le nom du fichier.

   Ici, si `find` sélectionne 10 fichiers, `grep` sera appelé 10 fois (une fois pour chacun des fichiers).

   Enfin, le `\;` est là pour marquer la fin de l'action `-exec`. Ce marqueur est obligatoire, et rend possible l'ajout d'autres actions après `-exec` (typiquement un autre `-exec`).

9. Consultez le man de `grep` et adaptez la commande précédente pour avoir en plus les lignes de code correspondantes avec leurs numéros.

Exercice 12 - Applications de `find`

1. Quel est le nombre de fichiers `.c` ou `.h` dans l'arborescence `/pub/FISE_OSSE11/shell/projet` ?

2. Combien de lignes de code font l'ensemble de ces fichiers ?

   note : Utilisez `cat` dans un `-exec`.

3. Combien de ces lignes de code contiennent la fonction (dépréciée) `atoi` ?

4. Faites une copie de l'arborescence `/pub/FISE_OSSE11/shell/projet` dans `/tmp/projet`.

```
sh$ rm -rfv /tmp/projet
sh$ cp -rv /pub/FISE_OSSE11/shell/projet /tmp/projet
```

   a. Faites une sauvegarde (`.c.orig` et `.h.orig`) de tous les fichiers `.c` et `.h` de l'arborescence `/tmp/projet`.
   b. Vérifiez qu'il y a autant de fichiers `.orig` que de fichiers `.c` et `.h`.
   c. Changez dans tous les `.c` et les `.h` de l'arborescence `/tmp/projet` les occurrences de `dupont` par `Dupont` et de `jean` par `Jean`.

      aide :

      — La commande **sed -i -e 's/mot1/mot2/g' file** change dans le fichier `file` toutes les occurrences de `mot1` par `mot2` (`-i` = *in place*).
      — On peut utiliser plusieurs fois l'option `-e`.

   d. Vérifiez qu'il n'y a plus de fichier `.c` ou `.h` de l'arborescence qui contienne `jean` ou `dupont`.

```
sh$ find /tmp/projet -type f -exec grep -w -l -e jean -e dupont {} \;
```

   e. Restaurez les fichiers `.c` et les `.h` du dossier `/tmp/projet` qui contiennent `Jean` à partir des fichiers `.orig`.

      — Faites un `find` qui lance `grep -q` (mode silence) via `-exec` et se termine par `-print`.
      Le `-print` est exécuté si tous les prédicats précédents sont vrais, en particulier le `grep`.
      — Remplacez l'action `-print` par un nouveau `-exec` qui lance la copie avec `cp`.

   f. Recherchez les fichiers de l'arborescence qui contiennent `Jean`.

```
sh$ find /tmp/projet -type f -exec grep -w -l -e Jean {} \;
```

   Il ne devrait plus y en avoir.
   g. Recherchez les fichiers de l'arborescence qui contiennent `Dupont`.

```
sh$ find /tmp/projet -type f -exec grep -w -l -e Dupont {} \;
```

   Vous devez trouver seulement `liste.c` et `liste.h`.

5. Recopiez l'arborescence `/pub/FISE_OSSE11/shell/projet` dans `/tmp/proj` mais seulement les répertoires, les `.c` et les `.h`.

   — Reconstruisez d'abord l'arborescence des répertoires (utilisez `find` et `mkdir`).
   — Copiez les fichiers `.c` et `.h` (utilisez `find` et `cp`).

Exercice 13 - `tar`

La commande `tar` est à Unix ce que `Winzip` est à Windows : elle permet de créer et de manipuler des archives.

Par exemple, la commande ci-dessous crée l'archive `file.tar` contenant les `path`*i*. Ceux ci peuvent être des répertoires ou des fichiers de n'importe quel type.

tar -cf file.tar path1 path2 path3 ...

Ensuite, la commande

tar -xf file.tar

permet d'extraire tous les fichiers de l'archive `file.tar` dans le dossier courant. On peut aussi n'extraire que certains fichiers :

tar -xf file path1 path2

La commande `tar` dispose de plusieurs autres options :

`-C dir` pour utiliser le répertoire `dir` à la place du dossier courant,

`-v` pour activer le mode verbeux (et donc afficher davantage d'informations),

`-t` pour afficher le contenu de l'archive (sans extraire les fichiers),

`-z` **ou** `-j` couplé à `-c` pour en plus compresser les archives à leur création (aux formats `gzip` et `bzip2` respectivement).

Enfin, comme de nombreuses autres commandes, `tar` peut lire sur l'entrée standard ou écrire sur la sortie standard. Il suffit pour cela d'utiliser `-` comme nom de fichier. Ceci peut s'avérer pratique notamment pour envoyer des données compressées via le réseau sans avoir à créer localement une archive.

1. Créez une archive du répertoire `/pub/FISE_OSSE11/shell/projet` dans `/tmp/projet.tar`.

```
sh$ ( cd $(dirname /pub/FISE_OSSE11/shell/projet) ; tar -cvf /tmp/projet.tar projet )
```

   ou

```
sh$ tar -C $(dirname /pub/FISE_OSSE11/shell/projet) -cvf /tmp/projet.tar projet
```

2. Visualisez les noms des fichiers de l'archive :

```
sh$ tar -tf /tmp/projet.tar
```

   ou en mode verbeux :

```
sh$ tar -tvf /tmp/projet.tar
```

3. Restaurez l'archive dans `/tmp`.

```
sh$ tar -C /tmp -xf /tmp/projet.tar
```

4. Détruisez l'arborescence `/tmp/projet` et l'archive `/tmp/projet.tar`.

5. Grâce à `tar` et `find`, créez l'archive `/tmp/code.tar` contenant uniquement les fichiers `.c` et `.h` de l'arborescence `/pub/FISE_OSSE11/shell/projet`.

Exercice 14 - Commandes shell **(examen 2022-2023)**

Donnez, pour chacun des besoins suivants, une commande shell répondant à ce besoin :

1. supprimer le répertoire `/tmp/log` à condition qu'il soit vide,
2. afficher les 5 premières lignes du fichier `projet.c`,
3. afficher les lignes du fichier `main.ml` contenant le mot `Array`,
4. remplacer toutes les occurrences de `IBD` par `CBDR11` dans le fichier `maquette.html`,
5. calculer le nombre de sous-répertoires présents dans le HOME de l'utilisateur courant,
6. afficher la liste des utilisateurs ayant un fichier dans `tmp`, triée par ordre alphabétique.

   note : On rappelle que, sous Linux, le propriétaire du fichier `file` peut être obtenu grâce à la commande `stat -c '%U' file`.

Exercice 15 - Simplification de commandes **(examen 2017)**

Réécrivez les commandes ci-dessous de manière plus optimisée :

1. `cat < 1 | cat | cat > 2`
2. `x="1$(echo 2 3)1"`
3. `cat < $(echo 1)`
4. `cat "$(echo 1 > 2)2"`

Exercice 16 - Révisions

1. Compilez le fichier `/pub/FISE_OSSE11/shell/projet/src/util/bug.c` et déterminez la première erreur.

   aide : On paginera la sortie d'erreur de `gcc` avec `less`.

2. Copiez le fichier `/pub/FISE_OSSE11/shell/projet/src/include/data.h` dans votre HOME.

3. Y a-t-il dans l'arborescence `/pub/FISE_OSSE11/shell/projet` un fichier de taille nulle ?

4. Quelle est la taille du plus grand fichier `.c` de l'arborescence `/pub/FISE_OSSE11/shell/projet` ?

   aide : utilisez `find`, `stat`, `sort` et `head`.

5. Quels sont les noms des fichiers `.c` dont la taille est celle de la question précédente ?

6. Quels sont les utilisateurs qui ont des fichiers réguliers dans `/pub/FISE_OSSE11` ?

   aide : utilisez `find`, `stat` et `sort`.

7. Combien y a-t-il de fichiers (réguliers) dans `/pub/FISE_OSSE11` appartenant à `christophe.mouilleron` ?

8. Quelle est la taille cumulée en octets de ces fichiers ?

9. Déterminez les fichiers `.c` de l'arborescence `/pub/FISE_OSSE11/shell/projet` qui incluent le fichier `table.h` directement.

10. Créez une archive `/tmp/burger.tar` qui contiendra tous les fichiers (réguliers) de l'arborescence `/pub/FISE_OSSE11/shell/projet` écrits par Patrice Burger (on sélectionnera les fichiers contenant la chaîne `"Patrice Burger"`).

11. Affichez le fichier `/pub/FISE_OSSE11/shell/select.dat`.
    Réaffichez-le en écrivant les dates au format `"aaaa/mm/jj"`.

12. Affichez le fichier `/pub/FISE_OSSE11/shell/select.dat` en le triant par date croissante.

    aide : utilisez `sort -k` en plus de `sed`.

Exercice 17 - `/dev/pts/*`, `tty`, `stty`

1. Déterminez à l'aide de la commande `tty` le nom du fichier spécial associé à votre terminal.

```
sh$ tty
```

   Ce nom dépend de votre session en cours. Vous n'aurez donc potentiellement pas le même résultat que votre voisin.

2. Lancez un nouveau terminal nommé A.

```
sh$ xterm -name A &
```

3. Déterminez le fichier spécial correspondant au terminal A.

```
sh$ tty # dans le terminal A
```

   Dans la suite, on notera `/dev/pts/N-A` pour désigner le fichier ainsi obtenu.

4. Dans le terminal A, changez le comportement de sorte que le « retour-arrière » soit maintenant produit par la touche `m`, et que le signal de fin de fichier soit produit par la touche `f`.

```
sh$ stty erase m # dans le terminal A
sh$ stty eof f # dans le terminal A
```

5. Écrivez quelque chose dans le terminal A et essayez les touches « retour-arrière » et m.

6. Lancez la commande `cat` et indiquez la fin du flux stdin en tapant sur la touche `f`.

```
sh$ cat # dans le terminal A
aaa
aaa
f
```

7. Restaurez les fonctions d'effacement de caractère et de fin du flux stdin du terminal A à leurs valeurs initiales.

```
sh$ stty -g # dans le terminal A
sh$ stty -g # dans un autre terminal pour comparer
sh$ stty erase 0x7f # dans le terminal A
sh$ stty eof 0x04 # dans le terminal A
```

   Essayez la touche « retour-arrière » et le raccourci <CTRL-D> pour vérifier qu'ils ont bien été restaurés.

8. Lancez un nouveau terminal nommé B, et déterminez le fichier spécial correspondant, qu'on appellera `/dev/pts/N-B`.

9. Lancez une commande `cat` sur le terminal A de telle sorte que les caractères tapés dans A s'affichent sur le terminal B.

```
sh$ cat >/dev/pts/N-B
```

10. Indiquez à `cat` la fin de la saisie.

```
en tapant <CTRL-D> dans le terminal A
```

11. Lancez une commande `echo` sur le terminal B qui écrit "hello world" sur le terminal A.

```
sh$ echo 'hello world' > /dev/pts/N-A
```

12. Un intérêt de la commande `stty` est illustré par la suite de commandes :

```
sh$ stty -echo ; read -p 'Password: ' pass ; stty echo ; echo
```

    Expliquez dans le détail ce qui se passe, sachant que la documentation de `read` est fournie dans la page man de `sh`.