Ligne de commande

Exercice 1. Questions de cours Les questions de cours sont à destinées à vous permettre de vérifier votre compréhension du cours. Elles sont à travailler à l'avance et ne seront pas traitées en TD ou TP.

1. Quel est le rôle d'un système d'exploitation ?
2. Qu'est-ce que la multiprogrammation dans un système d'exploitation ?
3. Quelle est la différence entre une exception et une interruption dans un système ?

Exercice 2. Ligne de commande Le shell est l'interface de base permettant à un utilisateur d'interagir avec le système d'exploitation. Une grande quantité de commandes sonit en pratique directement transcrites sous forme d'appels système, notamment celles permettant la manipulation des fichiers et répertoires :

cd	Change le répertoire courant
ls	Liste le contenu d'un répertoire
mv	Déplace un fichier
cp	Copie un fichier
rm	Supprime un fichier
mkdir	Crée un répertoire
rmdir	Supprime un répertoire
chown	Change le propriétaire d'un fichier
chmod	Change les droits d'accès à un fichier
...	

Ces commandes sont documentées dans les pages de manuel des système UNIX qui sont accessibles à l'aide de la commande `man`. Dans un terminal, vous pouvez taper `man` suivi du nom d'une de ces commandes afin d'obtenir sa documentation complète.

1. Lisez rapidement la documentation des commandes de base que vous ne connaissez pas afin de vous faire une idée de ce qu'elles permettent de réaliser.

Exercice 3. Traitement de données avec le shell Mais beaucoup plus de commandes sont disponibles et le shell permet de connecter ces commandes de manière à réaliser des tâches potentiellement complexes. Par exemple, la commande `head` permet d'afficher les `n` premières lignes d'un fichier et la commande `tail` permet quant-à-elle d'afficher les `n` dernières lignes.

Le premier méchanisme offert est la redirection de l'affichage d'une commande dans un fichier, ce qui permet à l'aide des deux commandes précédentes d'obtenir par exemple les lignes du milieu d'un fichier. Les deux commande suivante affiche les lignes 15 à 23 d'un fichier :

```
head -n 23 fichier.txt > temp.txt
tail -n 9 temp.txt
```

Le symbole > indiquant que l'affichage de la première commande doit être écrit dans le fichier `temp.txt`.

La majorité de ces commandes prennent leur données d'entrées depuis un fichier mais si celui-ci n'est pas fourni, elle liront les données depuis le clavier. Le shell offre aussi un méchanisme permettant de rediriger l'entrée d'un programme depuis un fichier ce qui permet de simuler le cas ou l'utilisateur taperais au clavier le contenu du fichier. Par exemple, la commande tail de l'exemple précédent aurait pus être écrite :

```
tail -n 9 < temp.txt
```

Le symbole < indiquant que l'entrée de la commande doit être lue depuis le fichier `temp.txt` à la place du clavier.

Ces commandes deviennent particulièrement intéressantes lorsque l'on combine les deux méchanismes à l'aide de ce que l'on appel un *pipe* représenté par le symbole `|`. La séquence de deux commandes ci-dessus peut être réalisée de manière plus consise et efficace, et sans avoir besoin de fichier intermédiaire, de la façon suivante :

```
head -n 23 fichier.txt | tail -n 9
```

Le symbole | ici indique que les deux commande doivent être lancées en même temps et que les données en sortie de la première commande doivent être données en entrée de la seconde.

Le tableau suivant vous liste quelques'unes des commandes de manipulation de fichier disponible sur la majorité des système UNIX :

cat	affiche un fichier
more	affiche page par page un fichier
head	filtre les n premières lignes
tail	filtre les n dernières lignes
comm	filtre les lignes communes de deux fichiers
diff	affiche les différences entre deux fichiers
split	decoupe un fichier en plusieurs fichiers
paste	concatène les lignes de plusieurs fichiers
cut	filtre les colonnes d'un fichier en fonction d'un séparateur
tr	réalise des substitutions de charactères
uniq	supprime les lignes consécutive identiques
fold	limite la longueur maximale des lignes d'un fichier
grep	filtre les lignes contenant une certaine chaine de charactère
wc	compte le nombre de charactères/mots/lignes
sort	tri les lignes
join	combine les lignes d'un fichier en réalisant une jointure
nl	numérote les lignes d'un fichier
...	

Utilisez et combinez ces commandes afin de répondre aux question suivantes. Leur documentation est aussi disponible dans les pages de manuel de vos système. *(dans cet exercice, par simplicité, on considèrera qu'un mot est une séquence de caractère séparée par des espaces)*

1. Comment afficher le deuxième mot de chaque lignes d'un fichier ?

   Correction: *Différentes manières de le faire. Le plus simple est d'utiliser la commande* `cut`*, le fichier pouvant être donné directement ou via une redirection.*

   ```
   cut -d ' ' -f 2 fichier.txt
   cut -d ' ' -f 2 < fichier.txt
   ```

2. Comment obtenir une liste (sans doublons) de tous les mots présents dans un fichier ?

   Correction: *La combinaison de trois commande permet d'obtenir le résultat désiré :*

   ```
   tr ' ' '\n' < fichier.tex | sort | uniq
   ```

   La première commande `tr` *remplace tous les espaces par des retour à la ligne. Ce qui à pour effet de placer un mot sur chaque lignes. La deuxième commande tri la liste de mots ainsi obtenue placant donc tous les mots dupliqués consécutivement dans le fichier afin que les doublons puissent être suprimmés par la dernière commande.*

3. Comment obtenir une liste des 50 mots les plus fréquents présent dans un fichier ?

   Correction: *En partant de la réponse précédente on obtient :*

   ```
   tr ' ' '\n' < fichier.tex | sort | uniq -c | sort -nr | head -n 50
   ```

   On ajoute le paramètre `-c` *à la commande* `uniq` *afin que celle-ci ajoute les comptes en début de lignes. On trie ensuite le fichier selon l'ordre numérique en plaçant les plus grandes valeurs en premier et on ne garde que les 50 premières lignes du résultat.*

4. Comment afficher les lignes d'un fichier commençant par une majuscule ?

   Correction:

   ```
   grep -E "^[A-Z]" fichier.txt
   ```

Exercice 4. Un cas plus concret Pour gérer les secteurs d'activité d'entreprises, on dispose d'un fichier nommé `entreprise.txt`. Toute ligne de ce fichier a la structure suivante :
Afin de tester vos commandes, copier les lignes suivantes dans un fichier de test.

```
91,Le Bec Fin:Restauration
74,Mille Pages:Edition
94,Resto Ville:Restauration
91,Vite Transport:Transport
63,Mes Souvenirs:Edition
94,Chez Marcel:Restauration
```

1. Écrire une commande qui crée un fichier `restauration.txt` contenant toutes les lignes correspondant au secteur d'activité « Restauration ».
Correction:

```
grep ":Restauration$" entreprise.txt > restauration.txt
```

2. Écrire une commande qui affiche les lignes correspondant aux entreprises de restauration du Val de Marne. (département 94)
Correction: *Deux variantes possibles suivant si l'on réutilise le résultat de la quastion précédente ou pas.*

```
grep "^94," entreprises.txt | grep ":Restauration$"
grep "^94," restauration.txt
```

3. Écrire une commande qui affiche le nombre d'entreprises de l'Essone. (département 91)
Correction:

```
grep -c "^91," entreprise.txt
```

4. Écrire une commande qui affiche le nombre de secteurs d'activités différents contenus dans le fichier.
Correction:

```
cut -d ':' -f 2 | sort -u | wc -l
```

5. Écrire une commande qui affiche par ordre alphabétique les noms des entreprises dans le secteur de l'édition.
Correction:

```
grep ":Edition$" entreprise.txt | cut -d ',' -f 2 | cut -d ':' -f 1 | sort -u
```

Exercice 5. Premiers appels systèmes Sous Linux, ainsi que sous de nombreux autres système UNIX, il est possible de réaliser directement des appels système sans utiliser de code assembleur à l'aide de la fonction :

```
long syscall(long number, ...) ;
```

néanmoins son utilisation est fortement déconseillée car elle est peu portable et nécessite de connaître à la fois le numéro de l'appel système ainsi que le détail de ses arguments et que le compilateur ne peut effectuer aucune vérifications.

Afin d'utiliser plus simplement les appels systèmes, la *libc* fourni la grande majorité des appels sous la forme de fonction classique du C telles que :

```
pid_t getpid(void) ;
```

qui permet à un processus de connaitre le numéro qui l'identifie.

Dans certains cas, l'interface fournie par l'OS est peu pratique ou regroupe trop de fonctionalitées différentes et est donc complexe à utiliser. Pour beaucoup d'appels systèmes, la *libc* fourni alors différentes fonctions qui offre une interface plus agréable pour un appel système. Par exemple, un processus peut demander à être terminé à l'aide de l'appel système :

```
void _exit(int status) ;
```

Mais la *libc* fourni aussi la fonction :

```
void exit(int status) ;
```

qui elle aussi demande la terminaison du programme mais après avoir proprement fermé tous les fichiers ouverts afin d'éviter des pertes de données.

Afin d'utiliser ces appels système, il est nécessaire d'inclure le fichier d'en-tête `unistd.h` le plus souvent. Certains appels sont regroupés dans d'autre fichiers, la documentation de ces appels dans les pages de manuel vous indiquera quels fichiers inclure.

1. La fonction `sleep` permet à un programme de demander à être mis en pause pendant un certain temps. Lisez sa documentation et écrivez un programme qui patiente 3 secondes avant de se terminer.
Correction:

```
1 #include <stdio.h>
```

```
2 #include <unistd.h>
3
4 int main(void) {
5     printf("Bonjour\n");
6     sleep(3);
7     printf("Au revoir\n");
8     return 0;
9 }
10
```

2. Cette fonction `sleep` utilise en fait l'appel système `nanosleep` afin de mettre le processus en pause. Lisez sa documentation et Récrivez le programme afin d'utiliser directement l'appel système.

Correction: *L'appel système offre une plus grande précision et un meilleur contrôle du temps de pause au prix d'une interface plus complexe à utiliser dans les cas les plus simples.*

```
1 #include <stdio.h>
2 #include <time.h>
3
4 int main(void) {
5     struct timespec ts;
6     ts.tv_sec = 3;
7     ts.tv_nsec = 0;
8     printf("Bonjour\n");
9     nanosleep(&ts, NULL);
10    printf("Au revoir\n");
11    return 0;
12 }
13
```