A la découverte du Bash

Linux existe depuis longtemps… C'est sans doute ça qui explique que la **console** a gardée autant d'importance aujourd'hui. Vous vous en êtes déjà rendu compte, il faut bien souvent lâcher la souris et utiliser le clavier pour communiquer avec le système…

Pour que vous ayez une bonne connaissance du système Linux, il faut que vous appreniez quelques commandes Linux (et il existe souvent la même commande sous **UNIX et Macintoch**). Avant de voir ces commandes, il faut savoir que les minuscules et les majuscules ont de l'importance… Donc un répertoire Toto est un répertoire différent que toto…

Ensuite, il est possible de cumuler les arguments en séparant d'un espace entre chaque argument.

- `ls` : Vous l'avez rencontré, c'est pour lister le contenu du répertoire courant. Arguments :
  - –a : affiche tous les fichiers (all), y compris les fichiers cachés.
  - –l : affiche beaucoup de détails sur les fichiers (long).
  - –u : affiche les utilisateurs qui ont lancé l'application.
- `cd` : Utiliser pour changer de répertoire.
- `pwd` : permet d'afficher le chemin du répertoire dans lequel on est (Print Working Directory)
- `mkdir` : permet de créer un répertoire (le dernier argument doit être le nom du répertoire)
- `rm` : permet de supprimer un fichier(le dernier argument doit être le nom du fichier…)
  - –f : forcer la suppression du fichier
  - –R : supprimer récursivement le répertoire (un répertoire est considéré comme un fichier…). Cela permet de supprimer les sous répertoires.
- `find` : Comme vous pouvez l'imaginer, ça permet de rechercher un fichier.
  - –name "`lenom`" : permet de rechercher le fichier "`lenom`" dans le répertoire courant et tous les sous-répertoires.
- `g++` : permet de compiler un programme écrit en c++.
  - –o : permet de dire au compilateur de créer un fichier exécutable
  - –c : permet de dire au compilateur de ne faire que compiler le programme
- `man` : permet d'avoir de la documentation sur une commande
  - "`laCommande`" : permet d'afficher de l'aide sur la commande.
- `whereis` : permet de savoir dans quel répertoire se trouve une commande.

Il est important que vous testiez chaque commande, et que vous les **appreniez par cœur** ! Attendez-vous à des questions dessus le jour du devoir !

Exercice 1 :

Utilisez ces commandes pour faire les actions suivantes :

- créez un répertoire `toto` dans répertoire `/home/eleve/Bureau` (ou `/home/eleve/Desktop` suivant la version de linux que vous avez…)
- Donnez le répertoire où se trouve la commande `sudo`
- Donnez une explication de la commande `sudo`
- Supprimer le répertoire que vous avez créé

Notre bash à nous

En plus de ces commandes (qui sont en réalité des **programmes** écrit en C et qui se trouvent dans le répertoire `/bin/`), il faut savoir que le prompt (>_) est aussi un programme. Aujourd'hui, nous allons essayer de reproduire ce programme ! Nous n'allons pas essayer de reproduire toutes les fonctions du `bash` (c'est comme ça que ça s'appelle) comme les touches haut et bas qui permettent de rappeler des fonctions précédemment tapées.

Il faut donc réfléchir à la manière dont ce programme travaille :

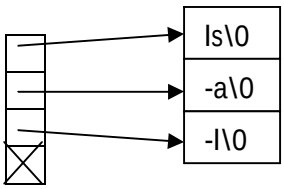
- Le prompt attend que l'utilisateur saisisse une commande (c'est juste une chaine de caractère)
- Quand l'utilisateur appuie sur entrée, le `bash` créé un nouveau processus pour exécuter la commande et attend le retour du processus.
- Quand la commande est finie, le programme attend une nouvelle commande et on revient à l'étape 1…

Deux choses importantes :

1) Il faut créer un processus fils à chaque fois que l'utilisateur a saisi une ligne de commande
2) Dans le processus fils, il faut utiliser `exec` pour exécuter la commande écrite.

Exercice 2 :

Pour pouvoir utiliser `exec`, il faut transformer la ligne de commande en tableau d'arguments. Par exemple, si l'utilisateur tape : `"ls –a –l"`, la commande `exec` attend comme premier argument `"ls"`, et comme deuxième argument un tableau de pointeurs sur une chaine de caractères. Dans cet exemple, ce tableau devra être de cette forme :



2 choses importantes : le tableau de pointeurs doit se terminer par une case à NULL, et chaque chaine de caractères doit se terminer par le caractère \0.

Faites une fonction qui prend en paramètre une chaine de caractère et qui retourne le nombre de paramètres que l'utilisateur a saisi. La signature de la fonction est la suivante :

```
int compteParam(string commande)
```

Pour faire ça, il faut utiliser la fonction find(" ",0) de la classe string. Elle permet de rechercher un espace dans la chaine de caractère de type string. Exemple :

```
string test("ls –a"); //on initialise test avec "ls –a"
test.find(" ",0); //renvoie 2 (on commence à 0)
test.find(" ",3);//renvoie string::npos parce qu'il n'y a plus
d'espaces après ce premier espace.
```

Grace à cette commande, trouvez le moyen de compter le nombre d'arguments…

Exercice 3 :

Maintenant que vous connaissez le nombre d'arguments, vous pouvez créer le tableau de pointeurs sur des chaines de caractères. Créer un tableau de tableau de caractère se fait comme ça :

```
char ** monTab=new char*[4] ;
```

Bien entendu, vous mettrez à la place de 4 le nombre d'arguments +1 (car le tableau **doit** se terminer par `NULL`, donc une case en plus…).

Pour chaque case, vous devrez faire un truc comme ça :

```
monTab[0]=new char[5] ;
```

avec à la place de 5, la taille du paramètre +1 (pour le `\0` à la fin), et dans chaque case une lettre…

Une fois que votre tableau est bien rempli, vous pouvez maintenant utiliser `exec` avec ce paramètre. Mais pour qu'après vous ayez la possibilité de ressaisir une commande, il faut utiliser `fork`() avant de faire `l'exec` dans le fils, et dans le père, il faut attendre le fils : `pid = wait(&status);` avec dans `status` le code de retour (0 si y'a pas eu d'erreurs…) et `pid` qui est le numéro du fils qui a fini…

Astuce :

Pour simplifier le code, je vous conseille de faire une fonction ProcessFils() et une fonction ProcessPere() pour bien séparer les deux implémentations. Le code devient alors beaucoup plus simple à comprendre :

```
if ((pid1 = fork()) < 0) {
          cout<<"Erreur dans le fork\n");
          exit(1);
     }
     else if (pid1 == 0)
          ProcessFils();
     else ProcessPere();
```

C'est dans le père que vous saisissez la commande, et c'est dans le fils que vous traitez la commande… Vous aurez sans doute à rajouter des paramètres aux deux fonctions, à vous de voir…