

## SOMMAIRE:

1. Listing du programme
  - a. Parsing()
  - b. Commande()
  - c. Main
2. Fonctionnement du programme
  - a. Parsing ()
  - b. Commande ()
  - c. Main
3. Test du programme
4. Difficultés rencontrées
5. Conclusion

BOUJEMAA Mariam

CENIK Selim

COPPIN Antoine

# INTRODUCTION

Dans le cadre du module programmation UNIX, nous avons été amenés à construire un mini Shell.

Notre Shell devait pouvoir réaliser certaines commandes. Ainsi, nous pouvons y exécuter des commandes faisant appel à des redirections ou encore à des tubes.

## 1. Listing du programme

### a. Parsing ()

```
void parsing(){
    int i=0;
    int cmot=0;
    while(1){
        c = getchar();
        if (c == '\n') {symboleP = 0;return;}
        else if (c == ';') {symboleP = 1;return;}
        else if (c == '&') {symboleP = 2;return;}
        else if (c == '<') {symboleP = 3;return;}
        else if (c == '>') {symboleP = 4;return;}
        else if (c == '|') {symboleP = 5;return;}
        else if (c == EOF) {symboleP = 7;return;}
        else if (c != ' ') {
            symboleP = 10;
            while(c != '\n' && !strchr(delimiteurs,c)){
                i=0;
                while(c != 32 ){
                    if((c != '\n') && !strchr(delimiteurs,c)){
                        mot[i]=c;i++;
                        c=getchar();
                    }
                    else {break;}
                }
                break;
            }
            while(c == ' ')
            {
                c=getchar();
            }
            ungetc(c,stdin);
            mot[i]=0;
            respP[cmot++]=strdup(mot);
            fflush(stdout);
            if(c == '\n' || strchr(delimiteurs,c))
            {
                respP[cmot]=0;
                return;
            }
        }
    }
}
```

## b. Commande ()

```
void commande () {
    pid_t pid, fid;
    int background = 0;
    int status;
    char car;
    int i, j, k, l;
    int p, p2;
    int execute=1;
    int output=0;
    int input=0;
    int tube=0;
    int fd[2];
    int fich;
    while(1){
        if(execute==1){
            if(symboleP==0){
                printf("DAUPHINE> ");
            }
            for (j=0;j<10;j++){
                respP[j]=NULL;
            }
            execute=0;
            background=0;
        }
        fflush(stdout);
        parsing();
        switch (symboleP){
            case 0 : // SYMBOLE : \n
                p=fork();

                if(p==0){ //fils
                    if(tube==1){//printf("\n\n\n");
                        fich = open("fichtmp",O_RDONLY,0640);
                        close(0); //fermeture clavier
                        dup(fich); //fichier devient entrée 0
                        execvp(respP[0], respP);
                        close(fich); //fermeture fichier
                    }
                    else if(output==0 && input==0){ //pas de redirection
                        printf("Erreur de redirection");
                        execvp(respP[0], respP);
                    }else if(output==1){ //dans le cas d'une redirection
                        printf("truc2");
                        close(1);
                        int filew = creat(respP[0], 0644);
                        execvp(respout[0], respout);
                    }
                    else if(input==1){
                        printf("truc3");
                        close(0);
                        int filer = open(respP[0], O_RDONLY);
                        execvp(respin[0], respin);
                    }
                    return 0;
                }else{ //pere
                    if(background==0){ //pas de bg on attend le fils
                        waitpid(p, NULL, 0);
                    }

                    output=0;
                    input=0;
                    execute=1;
                    tube=0;
                }
                break;
        }
    }
}
```

```

case 1:                                     // SYMBOLE : ;
    p=fork();
    if(p==0){                               //fils
        if(output==0 && input==0 && tube==0){ //pas de redirection
            execvp(respP[0], respP);
        }else if(output==1){                //dans le cas d'une redirection
            close(1);
            int filew = creat(respP[0], 0644);
            execvp(respout[0], respout);
        }
        else if(input==1){
            close(0);
            int filer = open(respP[0], O_RDONLY);
            execvp(respin[0], respin);
        }
        return 0;
    }else{                                  //pere
        waitpid(p, NULL, 0);
        output=0;
        input=0;
        execute=1;
    }
    break;

```

```

case 2:                                     // SYMBOLE : &
    background=1;
    break;
case 3:                                     // SYMBOLE : <
    if(input==0){
        input=1;
        execute=1;
        for (l=0;l<10;l++){
            respin[l]=respP[l];
        }
    }
    break;

```

```

case 4:                                     // SYMBOLE : >
    if(output==0){
        output=1;
        execute=1;
        for (l=0;l<10;l++){
            respout[l]=respP[l];
        }
    }
    break;

```

```

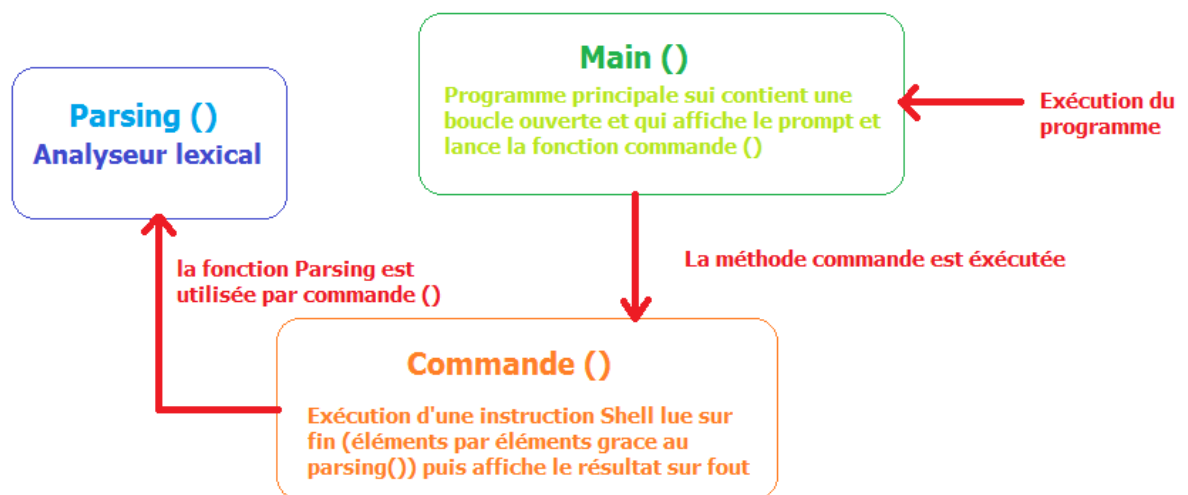
case 5:                                     // SYMBOLE : |
    //if(tube==0){
    /*for (l=0;l<10;l++){
        tube[l]=respP[l];
    }*/
    p2=fork();
    if(p2==0){
        if(tube==0){
            freopen( "fichier", "w", stdout );
            execvp(respP[0], respP);
        }
        return(0);
    }
    else{
        if(background==0){ // SANS MOD BG ATTENDRE FIN FILS
            waitpid(p2, NULL, 0);
        }
        tube=1;
        execute=1;
    }
    break;
default:

```

## c. Main

Liste des bibliothèques utilisées	Liste des variables globales	Programme main
<pre>#include &lt;unistd.h&gt; #include &lt;stdio.h&gt; #include &lt;sys/wait.h&gt; #include &lt;sys/stat.h&gt; #include &lt;fcntl.h&gt; #include &lt;sys/types.h&gt; #include &lt;stdlib.h&gt; #include &lt;string.h&gt; #include &lt;errno.h&gt;</pre>	<pre>char *delimiteurs = ";&amp;&lt;&gt; "; char *respP[20]; char *respout[10]; char *respin[10]; char *tube[10]; char mot[50]; int symboleP, status, c;</pre>	<pre>int main(int argc, char* argv[]) {     printf("Dauphine&gt;");     fflush(stdout);     while(1){         commande();     }     fflush(stdout); }  return 0; }</pre>

## 2. Fonctionnement de notre mini Shell



### a. Parsing

Le main fait appel à la fonction commande () qui lui-même fait appel à Parsing (). Nous choisissons de commencer par décrire cette commande car celle-ci n'appelle personne une fois appelée contrairement aux deux autres. Cette fonction est celle qui a été donnée nous ne l'avons pas modifiée.

Le Parsing lit le tableau respP qui va contenir une commande, cette commande sera inscrite par l'utilisateur, elle sera ensuite traitée à l'aide des délimiteurs.

Par exemple, si l'utilisateur choisi la commande `ls -l`, cette dernière sera automatiquement inscrite dans le tableau `respP []` et grâce aux délimiteurs, pris en compte par notre fonction `Parsing ()` elle sera traitée étape par étape.

**Exemple 1 :**

ls	-l								
----	----	--	--	--	--	--	--	--	--

**Exemple 2 :**

ls	;	ps							
----	---	----	--	--	--	--	--	--	--

C'est donc la valeur de retour de la fonction `Parsing ()` qui va nous indiquer quel est le délimiteur à traiter.

Par la suite, il faut traiter différemment la commande suivant les délimiteurs, en effet, si, comme dans l'exemple 2 on constate que le délimiteur est un `;` il faut le prendre en compte. La fonction fait donc appel à `commande ()` qui traite la commande en fonction des délimiteurs et exécute cette commande.

- La fonction `strdup` : duplique une chaîne de caractère et renvoie l'adresse de la nouvelle chaîne de caractère. (Fonction renvoie donc un pointeur soit `NULL` si il n'y a pas assez d'espace en mémoire ou sinon il renvoie l'adresse de la chaîne qui a été dupliqué).
- La fonction `ungetc` retourne un caractère vers un flux, si le caractère n'a pas été lu il renvoi EOF.

## b. Commande ()

Afin de répondre à la demande de l'utilisateur et traiter sa commande il suffit d'une part de créer un switch de commande qui traitera au mieux les différents délimiteurs repéré par notre fonction `Parsing ()`.

**Case 0 : \n EOF**

Retour à la ligne.

Il correspond à notre plus long cas car c'est

1. On commence avec le cas d'un pipe.

C'est à dire que le cas 5 nous a retourné qu'il y a bien un tube qui a été créé, ce qui est traduit par `tube = 1`. Il s'ensuit ensuite la procédure suivante:

2. S'il n'y a pas de pipe:

On exécute simplement le code.

A la fin de l'exécution normale du fils, on retourne vers le processus père, le `return 0` permet d'afficher le prompt `Dauphine>` conformément au Switch.

3. Dans le cas d'une non redirection, on affiche le message d'erreur : erreur de redirection et on lance de nouveau le prompt `Dauphine` avec le tableau `respP` qui va contenir la nouvelle commande d'exécution du Shell et le pointeur `rep` qui renvoie vers l'adresse de la commande lue.

**Case 1 : ;**

Commande est terminée, il faut donc l'exécuter grâce à `execv`, il s'agit d'abord de faire un `fork` pour lancer un processus fils qui exécutera la fonction. En effet, le point

#### Case 2 : &

& signifie que la commande peut être lancée et au même moment l'utilisateur a toujours la main afin d'exécuter d'autres commandes. Cela signifie que l'on ne doit pas attendre la fin de l'exécution du processus fils (`wait`) et directement passer la suite des commandes lancées par l'utilisateur.

#### Case 3 : <

L'entrée de la commande doit être modifiée. Il suffit de modifier l'entrée standard :

❶ La variable **input** va nous permettre de mener à bien notre case.

#### Case 4 : >

C'est le même cas que le case 3, il faut uniquement modifier la sortie standard :

- La variable **output** va nous permettre de mener à bien notre case.

#### Case 5 : |

Il suffit de récupérer le résultat de la commande dans un fichier. Puis de modifier l'entrée standard (Tout comme <). En passant en paramètre d'open le fichier qui sera créé pour effectuer toute les redirections de commande.

D'autre part, la commande est exécutée lors du cas 0 (new line), cas 1 (point-virgule) ou du cas 5 (pipe) par un `exec` sur le tableau qu'on lui a confié

Après avoir exécutée la commande, on sort du case, la commande est exécutée, on revient au main qui réaffiche automatiquement le prompt.

## c. Main ()

Le main fait appel à la fonction `commande()` qui lui même fait appel à `parsing()`.

A l'entrée du main, nous entrons une boucle afin d'afficher continuellement **Dauphine>** et appeler la fonction **commande()**.

### 3. Test du programme

On exécute un certain nombre de commande afin de tester la validité de notre programme

#### 1/ Test d'une commande simple

On traitera d'abord le cas d'une commande simple, par exemple `ls -l`.

```
DAUPHINE> ls -l
total 56
-rwxrwxr-x 1 ubuntu ubuntu 13751 janv.  9 23:35 c
-rw-rw-r-- 1 ubuntu ubuntu  7078 janv.  9 23:34 c.c
-rw-rw-r-- 1 ubuntu ubuntu  7080 janv.  9 23:34 c.c~
drwxr-xr-x 2 ubuntu ubuntu   80 janv.  9 22:57 Desktop
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Documents
-rwxrwxr-x 1 ubuntu ubuntu 13765 janv.  9 23:39 final
-rw-rw-r-- 1 ubuntu ubuntu  7163 janv.  9 23:35 final.c
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Images
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Modèles
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Musique
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Public
drwxr-xr-x 2 ubuntu ubuntu  100 janv.  9 23:36 Téléchargements
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Vidéos
```

#### 2/ Test d'une commande avec un délimiteur

Délimiteur '**;**'

**ls -l; ps**

Cette dernière affiche les deux commandes à la suite. En effet, l'utilisation du délimiteur **;** va permettre l'affichage de tous les fichiers du repertoire utilisateur avec les détails sur chaque fichiers ou repertoire (taille, droit ..) Puis en deuxième affichage la liste des processus du terminal.

```
DAUPHINE> ls -l;ps
total 60
-rwxrwxr-x 1 ubuntu ubuntu 13751 janv.  9 23:35 c
-rw-rw-r-- 1 ubuntu ubuntu  7078 janv.  9 23:34 c.c
-rw-rw-r-- 1 ubuntu ubuntu  7080 janv.  9 23:34 c.c~
drwxr-xr-x 2 ubuntu ubuntu   80 janv.  9 22:57 Desktop
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Documents
-rw-r--r-- 1 ubuntu ubuntu   114 janv.  9 23:42 fich
-rwxrwxr-x 1 ubuntu ubuntu 13765 janv.  9 23:39 final
-rw-rw-r-- 1 ubuntu ubuntu  7163 janv.  9 23:35 final.c
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Images
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Modèles
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Musique
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Public
drwxr-xr-x 2 ubuntu ubuntu  100 janv.  9 23:36 Téléchargements
drwxr-xr-x 2 ubuntu ubuntu   40 janv.  9 22:57 Vidéos
  PID TTY          TIME CMD
 5175 pts/1        00:00:00 bash
 5360 pts/1        00:00:00 final
 5406 pts/1        00:00:00 ps
```



### 3/ Test de la deuxième commande : ls -l | wc -l

```
ubuntu@ubuntu:~$ ls
1  c.c~      fich      final     Images    po        Téléchargements
c  Desktop  fichier  final.c   Modèles   pro.c     Vidéos
c.c Documents  fichtmp  final.c~  Musique   Public
ubuntu@ubuntu:~$ ls | wc -l
20
ubuntu@ubuntu:~$ ls | wc -c
141
```

La commande ls affiche les  
fichiers et sous rep du  
repertoire courant user

la commande présente un délimiteur avec un tube qui affiche le nombre de ligne présent dans le répertoire courant de l'user: on compte 20 fichiers

On affiche le nombre de caractères présent dans le même repertoire cette fois-ci.  
On compte 141 caractères.

#### 4/ Test de la redirection avec la commande : ps > fich

```
DAUPHINE> ps > fich
DAUPHINE> ps
```

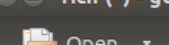
PID	TTY	TIME	CMD
5175	pts/1	00:00:00	bash
5360	pts/1	00:00:00	final
5378	pts/1	00:00:00	ps

Dans une première commande, on lance :

```
ps > fich
```

Cette commande renvoie dans le fichier fich la liste des processus utilisé dans le terminal.

```
PID
5175
5360
5375
fich
```



```

fich (~) - gedit
Open Save
c.c x final.c x fich x
PID TTY          TIME CMD
5175 pts/1          00:00:00 bash
5360 pts/1          00:00:00 final
5375 pts/1          00:00:00 ps

```

## 5/ On teste la commande grep sur le fichier mariam.txt

```
ubuntu@ubuntu:~$ grep '^$' mariam.txt | wc -l
4
```

maria  
maria  
jbfd

mariam.txt

Cette commande extrait le nombre de ligne vide du fichier mariam.txt puis renvoie le nombre de ligne vide

Afin de vérifier le résultat du pipe correspondant à 4 lignes vide, on ouvre le fichier et on compte par nous même le nombre de lignes vide.

On compte 4  
lignes vide

[illegible]

## 6/ Test de la redirection & du tube avec la commande : `ls -l | wc -l > fichier`

total  
-rw-r--r--  
-rw-r--r--  
-rw-r--r--  
fichier

final.c x fichier x  
21

```
ubuntu@ubuntu:~$ ls -l
total 120
-rwxrwxr-x 1 ubuntu ubuntu 13749 janv. 10 02:03 1
-rwxrwxr-x 1 ubuntu ubuntu 13751 janv. 9 23:35 c
-rw-rw-r-- 1 ubuntu ubuntu 7078 janv. 9 23:34 c.c
-rw-rw-r-- 1 ubuntu ubuntu 7080 janv. 9 23:34 c.c~
drwxr-xr-x 2 ubuntu ubuntu 80 janv. 9 22:57 Desktop
drwxr-xr-x 2 ubuntu ubuntu 40 janv. 9 22:57 Documents
-rw-r--r-- 1 ubuntu ubuntu 114 janv. 9 23:42 fich
-rw-rw-r-- 1 ubuntu ubuntu 3 janv. 10 02:05 fichier
-rw-rw-r-- 1 ubuntu ubuntu 1061 janv. 10 00:52 fichtmp
-rwxrwxr-x 1 ubuntu ubuntu 13765 janv. 9 23:39 final
-rw-rw-r-- 1 ubuntu ubuntu 7076 janv. 10 02:03 final.c
-rw-rw-r-- 1 ubuntu ubuntu 7083 janv. 10 02:03 final.c~
drwxr-xr-x 2 ubuntu ubuntu 40 janv. 9 22:57 Images
drwxr-xr-x 2 ubuntu ubuntu 40 janv. 9 22:57 Modèles
drwxr-xr-x 2 ubuntu ubuntu 40 janv. 9 22:57 Musique
-rwxrwxr-x 1 ubuntu ubuntu 13736 janv. 10 00:16 po
-rw-rw-r-- 1 ubuntu ubuntu 8243 janv. 10 00:13 pro.c
drwxr-xr-x 2 ubuntu ubuntu 40 janv. 9 22:57 Public
drwxr-xr-x 2 ubuntu ubuntu 120 janv. 10 00:13 Téléchargements
drwxr-xr-x 2 ubuntu ubuntu 40 janv. 9 22:57 Vidéos
```

```
ubuntu@ubuntu:~$ ls -l | wc -l
21
ubuntu@ubuntu:~$ ls -l | wc -l > fichier
```

On exécute la commande  
**ls -l**

Cette dernière affiche la  
liste de tous les fichiers/  
repertoires présent dans le  
repertoire courant.  
On compte alors 21  
fichiers.

En effet on ajoute l'exécution en  
séquence: **ls -l | wc -l**  
On compte alors 21 fichiers

Commande qui contient:  
redirection et tube.  
Cela consiste à mettre le résultat des  
deux premières commandes dans le  
fichier.

## 4. Difficultés rencontrées



## 5. Conclusion

Tout au long de la préparation de notre projet, nous avons pu appréhender la gestion d'une équipe, mais surtout du temps avec tous les impératifs et les imprévus que cela entraîne.

Pour commencer, ce projet nous a donné un premier aperçu des enjeux du travail en équipe en effet cela nous a permis de s'identifier à de jeunes informaticiens travaillant main dans la main dans des bureaux de grandes entreprises. Car, afin de réaliser les objectifs définis en début de projet qui consistait avant tout à créer un mini Shell en utilisant le langage C puis d'établir par la suite un compte rendu, il a fallu établir une véritable organisation c'est à dire une répartition des différentes tâches afin de se partager équitablement le travail.

Après la gestion de notre binôme, celle du temps fut primordiale également. Afin d'en perdre le moins possible, nous avons opté pour une présentation très concise car un travail à trois nécessite impérativement d'un codage riche en commentaire afin que l'on puisse facilement se relire et utiliser avec aisance son propre travail. À savoir que nous avons opté pour l'utilisation de Google drive, nous permettant ainsi de rester en contact malgré la distance qui nous sépare.

Enfin, le projet nous a permis d'appliquer les connaissances qui nous ont été inculquées au cours et aussi indispensable d'apprendre à travailler en équipe et à gérer notre temps. Nous avons été confrontés à de nombreux problèmes et dans la plupart des cas nous avons pu trouver une solution alternative afin de les résoudre partiellement par le biais d'internet à l'aide de différents tutoriels ou bien de manuel dédié à l'informatique.