

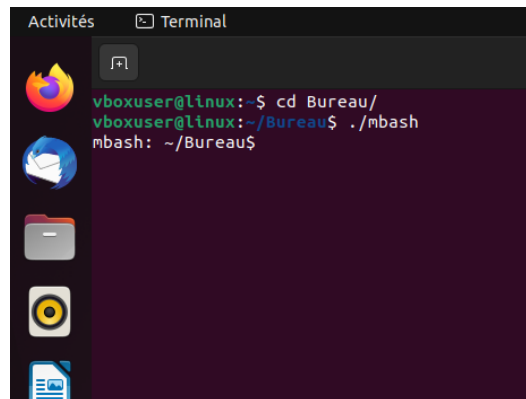
## **SAE : Réseau et application serveur**

### **I. mbash : une version miniature de bash :**

Le projet consiste à créer un bash personnalisé appelé "mbash" en C. Il inclut des fonctionnalités telles que la gestion des commandes basiques (ls, touch, pwd, ... ) ainsi que leur options (ls -l, gedit file.c &, gcc, ...). De plus, la commande history permet d'accéder à l'historique des commandes du mbash. Le programme permet aussi l'affichage du répertoire courant et la gestion des pipes.

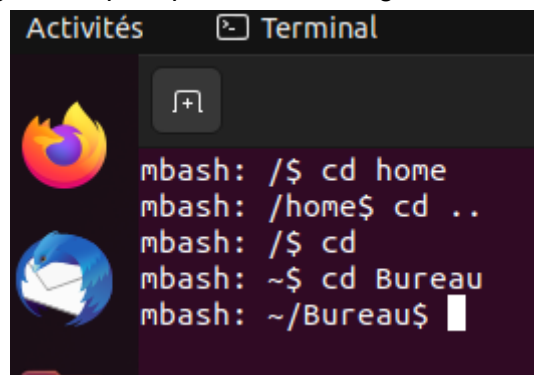
Notre mbash utilise la méthode `execvp()` pour exécuter les commandes reçues car elle cherche la librairie où se situe l'exécutable de la commande contrairement à `execve()`. Cela remplace le processus courant par un nouveau processus donc nous avons utilisé la fonction `fork()` pour créer un processus fils dédié à l'exécution. Il gère également le caractère "&" pour lancer des commandes en arrière-plan.

A chaque attente de commande on affiche le répertoire courant tout en remplaçant le chemin du home de l'utilisateur par "~" :



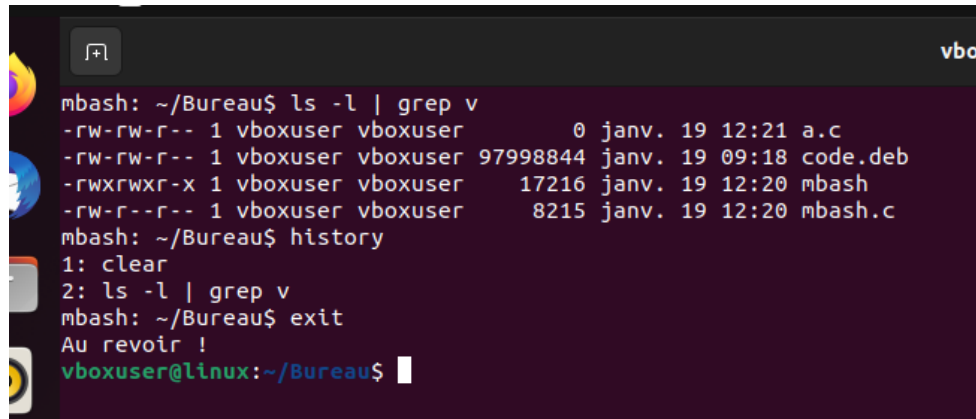
```
Activités Terminal
vboxuser@linux:~$ cd Bureau/
vboxuser@linux:~/Bureau$ ./mbash
mbash: ~/Bureau$
```

Le programme peut prendre en charge la commande `cd` :



```
Activités Terminal
mbash: /$ cd home
mbash: /home$ cd ..
mbash: /$ cd
mbash: ~$ cd Bureau
mbash: ~/Bureau$
```

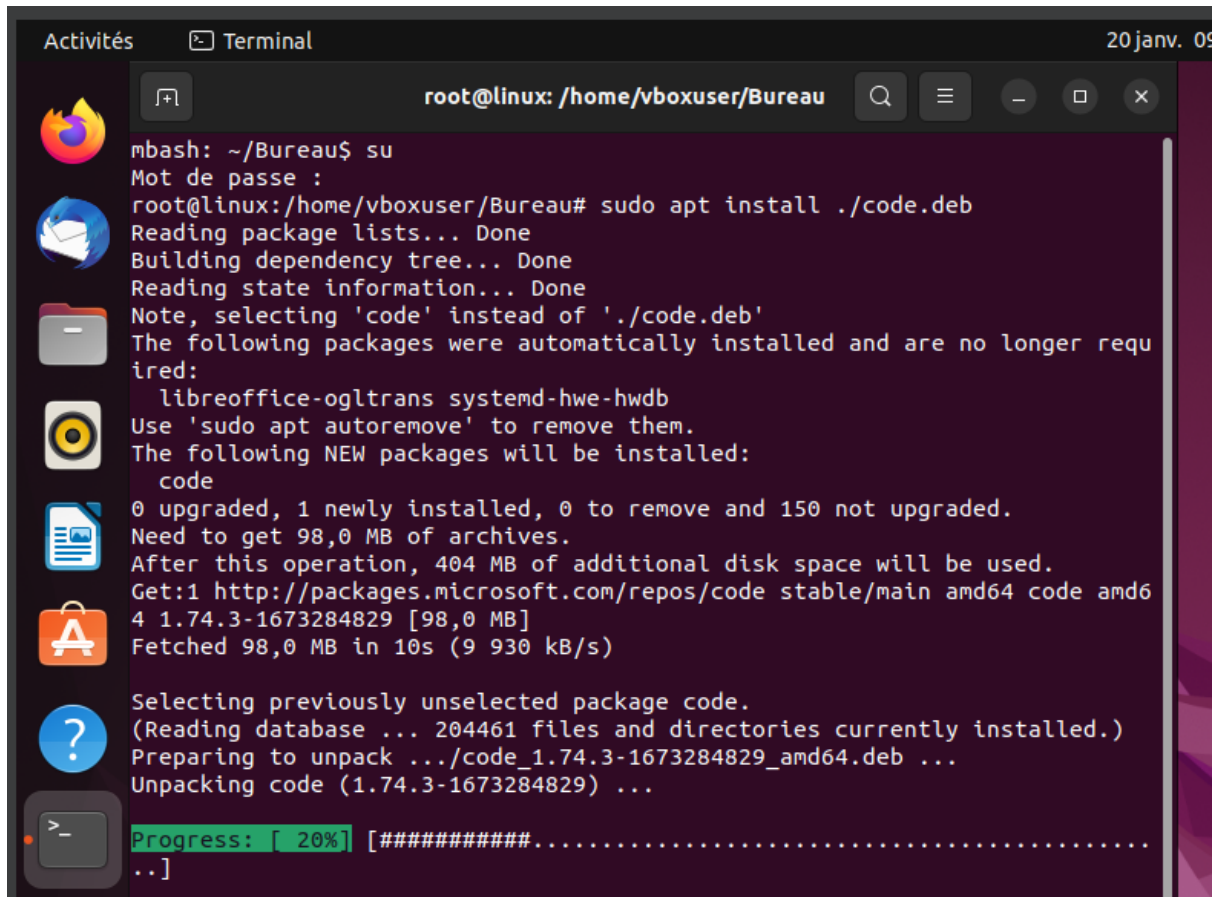
De plus, nous avons g rer des commande sp ciale telles que clear, exit, ctrl-D, history :



```
mbash: ~/Bureau$ ls -l | grep v
-rw-rw-r-- 1 vboxuser vboxuser      0 janv. 19 12:21 a.c
-rw-rw-r-- 1 vboxuser vboxuser 97998844 janv. 19 09:18 code.deb
-rwxrwxr-x 1 vboxuser vboxuser   17216 janv. 19 12:20 mbash
-rw-r--r-- 1 vboxuser vboxuser    8215 janv. 19 12:20 mbash.c
mbash: ~/Bureau$ history
1: clear
2: ls -l | grep v
mbash: ~/Bureau$ exit
Au revoir !
vboxuser@linux:~/Bureau$
```

Le ctrl-D est g r  avec fgets() car quand il retourne NULL cela vient g n ralement du ctrl-D donc nous arr tons la boucle   ce moment. Ensuite, pour l'historique des commandes nous avons d clar  un tableau de cha ne de caract re pour garder en m moire les commandes. Nous avons cr   les m thodes save\_history() qui ajoute dans le tableau la commande en param tre et d cale tout le tableau si on a atteint la limite. Ainsi que show\_history() pour afficher chaque commande.

De plus, on peut utiliser la commande su et installer/d installer des programmes :



```
root@linux: /home/vboxuser/Bureau
mbash: ~/Bureau$ su
Mot de passe :
root@linux:/home/vboxuser/Bureau# sudo apt install ./code.deb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'code' instead of './code.deb'
The following packages were automatically installed and are no longer required:
  libreoffice-ogltrans systemd-hwe-hwdb
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  code
0 upgraded, 1 newly installed, 0 to remove and 150 not upgraded.
Need to get 98,0 MB of archives.
After this operation, 404 MB of additional disk space will be used.
Get:1 http://packages.microsoft.com/repos/code stable/main amd64 code amd64 1.74.3-1673284829 [98,0 MB]
Fetched 98,0 MB in 10s (9 930 kB/s)

Selecting previously unselected package code.
(Reading database ... 204461 files and directories currently installed.)
Preparing to unpack .../code_1.74.3-1673284829_amd64.deb ...
Unpacking code (1.74.3-1673284829) ...
Progress: [ 20%] [#####.....]
..]
```

Enfin, pour les pipes, lorsque nous avons un “|” dans la commande elle est découpée en un tableau de chaîne de caractère. Ce dernier est envoyé à la méthode `pipe_mbash()` qui les fonctions `pipe()`, `close()`, `dup2()` et `fork()` pour rediriger la sortie des premières commandes aux suivantes.

En revanche, nous avons tenté d’implémenter les commandes pour la tabulation (auto-complétion), flèches directionnelles (naviguer dans l’historique, déplacer le curseur de la chaîne) et le `ctrl-R`. Malheureusement, nous n’arrivons pas à correctement récupérer leurs signaux sans paralyser le programme. Nous avons donc abandonné ces fonctionnalités.

## II. Serveur de package Debian :

### 1. Partie serveur :

On souhaite pouvoir mettre notre application dans un repository qui permet aux utilisateurs de télécharger notre application et de faire des mise à jour de celle-ci.

- 1) On va d’abord créer un paquet Debian signé avec une clé.
- a) On crée la structure du package :

```
mbash/ (le nom du package)
├── DEBIAN
│   ├── control (le fichier de configuration du package)
│   ├── postinst (script de post-installation)
│   ├── postrm (script de post-désinstallation)
│   ├── preinst (script de pré-installation)
│   └── prerm (script de pré-désinstallation)
│       (les scripts ne sont pas obligatoire)
└── usr
    └── bin
        └── mbash (notre programme mbash compilé)
```

- b) Dans le fichier `mbash/DEBIAN/control`, on va mettre les informations suivantes :

**Package:** mbash

**Version:** 1.0.0

**Section:** base

**Priority:** optional

**Architecture:** amd64

**Maintainer:** AlexEtZhiSheng

**Description:** Notre programme du mini-bash qui reproduit un bash mais en moins bien

- c) On peut maintenant créer notre package Debian avec la commande :

```
dpkg-deb --build mbash
```

Qui va créer le fichier **mbash.deb**

- 2) On va ensuite utiliser une clé GPG qui permettra de signer le paquet

- a) On génère la clé avec la commande : **gpg --gen-key**

La boîte de dialogue vous demandera d'entrer votre **nom**, **email** et un **mot de passe** qui est utile à déverrouiller la clé.

- b) On exporte la clé publique pour pouvoir la partager plus tard avec les utilisateurs :

```
gpg --export --armor [email] > public.key
```

On a donc la clé public sous forme de texte dans le fichier **public.key**

- c) On va ajouter la clé publique à notre trousseau de clé :

```
gpg --import public.key
```

On peut voir la liste de ses clé avec la commande : **gpg --list-key**

- d) On peut ensuite signé notre package : **dpkg-sig --sign builder mbash.deb**

3) Maintenant que le paquet est signé, on va le mettre dans un repository sur un serveur apache.

- a) On doit d'abord créer la structure de notre repository, que l'on va faire dans le répertoire **/var/www** du localhost d'apache :

```
mkdir /var/www/monRepo
```

```
cd /var/www/monRepo
```

```
mkdir -p {conf,incoming}
```

- b) Dans le dossier **conf**, on va ajouter un fichier qui contient les fichiers de configurations

```
touch conf/distributions
```

```
cat conf/distributions
```

```
Origin: mbash
```

```
Label: mbash
```

```
Suite: stable
```

```
Codename: mbash_1.0.0
```

```
Architectures: amd64
```

```
Components: main
```

```
Description: Apt repository for project mbash
```

- c) On met notre paquet mbash.deb dans le dossier incoming  
`mv mbash.deb /var/www/monRepo`
- d) On met la clé publique dans le dépôt : par exemple dans  
`/var/www/monRepo/key/public.key`

On doit maintenant avoir quelque chose comme ca :

```
monRepo
├── conf
│   └── distributions
├── incoming
│   └── mbash.deb
└── key
    └── public.key
```

- e) On va générer le reste des fichiers du repository avec la commande `reprepro`  
`reprepro includedeb stable monRepo/incoming/mbash.deb`  
et  
`reprepro export` pour mettre à jour la liste des paquets dans le dépôt.

## 2. Partie Client :

On va pouvoir maintenant installer notre mbash, côté utilisateur.

Sur la machine utilisateur du même réseau, on va faire les étapes suivantes.

- 1) On ajoute le repository en ligne au fichier de configuration des dépôts apt :  
`sudo nano /etc/apt/sources.list`
- 2) On ajoute la ligne suivante dans ce fichier :  
`deb http://localhost/monRepo stretch main`
- 3) On ajoute la clé GPG pour notre dépôt en utilisant la commande :  
`wget -q -O - http://localhost/monRepo/key/public.key | sudo apt-key add -`
- 4) On met à jour la liste des paquets en utilisant la commande :  
`sudo apt-get update`
- 5) On peut désormais installer l'application avec la commande :  
`sudo apt install mbash`
- 6) On peut désinstaller l'application avec la commande : `sudo apt remove mbash`

### 3. Cycle de vie :

Imaginons que l'on souhaite publier une nouvelle version de mbash.

#### Côté serveur :

- 1) On va refaire un nouveau package debian avec la nouvelle version de mbash.  
(suivre les étapes de création d'un package debian et de signature)
- 2) On va ajouter le nouveau package à notre repository :  
`reprepro includedeb stable monRepo/incoming/mbash.deb`  
puis  
`reprepro export`

#### Côté client :

On souhaite avoir la dernière version de mbash

On exécute la commande : `sudo apt-get update`