

SAE 3.03 - Réseau et application serveur

Sommaire

| | |
|--|----------|
| 1 - mbash : une version miniature de bash | 2 |
| 1 - Fonctionnalités | 2 |
| 2 - Code | 2 |
| 2 - Serveur de Package Debian | 5 |

1 - mbash : une version miniature de bash

1 - Fonctionnalités

- Commandes cd et pwd implémentées.
- Prise en compte des arguments avant d'exécuter les autres commandes (par exemple, ls -l fonctionne et prend en compte le -l).
- Prise en compte du symbole & en fin de commande pour l'exécuter en arrière-plan (facilement testable avec sleep 1 / sleep 1&).
- Commande exit pour quitter mbash.
- Commande history pour voir l'historique des commandes.
- Commande ! pour accéder à l'historique (avec !- fonctionnel). Si on rappelle une autre commande d'historique, on déroule jusqu'à 100 fois ; sinon, on peut avoir des boucles infinies (si la dernière commande est !-1 et qu'on la rappelle, elle va se rappeler en boucle).
- AVEC READLINE : autocomplétion sur les données de l'historique.
- AVEC READLINE : flèches haut et bas pour naviguer dans l'historique.
- On peut echo les variables d'environnement (comme echo \$HOME).
- On peut enchaîner les commandes avec ;.

À ajouter :

- Écrire la logique (genre while, if, then, etc.).
- Corriger le bug lorsqu'on appuie beaucoup sur tab.
- Pour écrire dans un fichier : >, 1>, 2>.

POUR COMPILER :

gcc -o mbash mbash.c -lreadline

Installer readline :

sudo apt-get install libreadline-dev

2 - Code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <readline/readline.h>
#include <readline/history.h>
#include <ctype.h>

#define MAX_CMD_LEN 1024
#define MAX_HISTORY 100

char* history[MAX_HISTORY];
int history_count = 0;

//ajouter une commande à l'historique
void add_to_history(char* cmd) {
    if (history_count < MAX_HISTORY) {
        //si l'historique n'est pas plein on ajoute simplement la commande
        history[history_count++] = strdup(cmd);
    } else {
        //sinon on libere la memoire de la premiere commande et on decale les autres
        free(history[0]);
        for (int i = 1; i < MAX_HISTORY; i++) {
            history[i - 1] = history[i];
        }
        history[MAX_HISTORY - 1] = strdup(cmd);
    }
}

//afficher l'historique des commandes
void print_history() {
    for (int i = 0; i < history_count; i++) {
        printf("%d %s\n", i + 1, history[i]);
    }
}

//remplacer les variables d'environnement dans une chaîne
char* replace_env_variables(char* cmd) {
    //allouer de la memoire pour le resultat
    char* result = malloc(MAX_CMD_LEN);
    char* p = cmd; //pointeur pour parcourir la chaîne d'entree
    char* r = result; //pointeur pour construire la chaîne de résultat

    //parcourir chaque caractere de la chaîne d'entree
    while (*p) {
        //si le caractere est un '$' -> variable d'environnement
        if (*p == '$') {
            p++; //skip le '$'
            char var_name[100]; //tableau pour stocker le nom de la variable
            char* v = var_name; //pointeur pour construire le nom de la variable
```

```

        //lire le nom de la variable (lettres, chiffres, ou '_')
        //isalnum sert a verifier si le caractere est une lettre ou un chiffre
        while (*p && (isalnum(*p) || *p == '_')) {
            *v++ = *p++;
        }
        *v = '\0';

        //on a le nom donc on recup avec getenv
        char* var_value = getenv(var_name);
        if (var_value) {
            //copier la valeur de la variable dans le resultat
            while (*var_value) {
                *r++ = *var_value++;
            }
        }
        else {
            //copier le caractere courant dans le resultat
            *r++ = *p++;
        }
    }
    *r = '\0';
    return result;
}

//fonction pour cd et pwd et history
void execute_interne(char* cmd) {
    if (strncmp(cmd, "cd", 2) == 0) {
        //on rentre ici si cmd est égal à "cd"
        char* dir = cmd + 3;
        //on récupère le répertoire car la commande est rentre comme ca: cmd = "cd /home/user"
        //donc on coupe les 3 premiers char
        //puis on change de repertoire avec chdir
        chdir(dir);
    }
    else if (strncmp(cmd, "pwd", 3) == 0) {
        //on prepare un tab pour recup le repertoire
        char cwd[MAX_CMD_LEN];
        if (getcwd(cwd, sizeof(cwd)) != NULL) {
            //on le print si il est pas null
            printf("%s\n", cwd);
        }
    }
    else if (strncmp(cmd, "history", 7) == 0) {
        print_history();
    }
    else if (strncmp(cmd, "echo", 4) == 0) {
        char* message = replace_env_variables(cmd + 5);
        printf("%s\n", message);
        free(message);
    }
}

//fonction pour les commandes externes
void execute_externe(char* cmd) {
    pid_t pid = fork(); //cree un nouveau processus pour executer la commande

```

```

if (pid == 0) {
    //ici on est l'enfant
    //compter les arguments
    char* temp_cmd = strdup(cmd); //pour ne pas modifier l'original on duplique
    int count = 0;
    char* pch = strtok(temp_cmd, " ");
    while (pch != NULL) {
        count++;
        pch = strtok(NULL, " ");
    }
    free(temp_cmd); //on libere la memoire

    //creer le tableau des args
    char* args[count + 1]; //+1 pour le NULL final !!
    int i = 0;
    pch = strtok(cmd, " ");
    while (pch != NULL) {
        args[i++] = pch;
        pch = strtok(NULL, " ");
    }
    args[i] = NULL; //NULL final

    //on execute la commande
    execvp(args[0], args);
    perror("execvp"); // Si execvp échoue
    exit(1);
} else if (pid < 0) {
    perror("fork"); //si le fork echoue on met une erreur
} else {
    //ici on est le parent et on attend que l'enfant finisse
    wait(NULL);
}
}

//fonction de completion avec tab avec les trucs de l'historique
//notation pour readline
char* command_generator(const char* text, int state) {
    static int list_index, len;
    char* name;

    //on verifie si on est au premier appelle
    if (!state) {
        //debut de l'index et de la longueur du texte a completer
        list_index = 0;
        len = strlen(text);
    }

    //parcourir l'historique des commandes
    while ((name = history[list_index++])) {
        //si le debut de la commande correspond au texte a completer
        if (strncmp(name, text, len) == 0) {

```

```

        //retourner une copie de la commande correspondante
        return strdup(name);
    }
}

//si aucune commande ne correspond, retourner NULL et on fait rien
return NULL;
}

//fonction de completion de commande
char** command_completion(const char* text, int start, int end) {
    rl_attempted_completion_over = 1; //pour dire qu'on a fini de completer
    return rl_completion_matches(text, command_generator); //readline va appeler
    command_generator pour completer
}

void execute_single_command(char* input){
    int bg = 0;
    if (input[strlen(input) - 1] == '&') {
        bg = 1; //determiner si la commande doit etre executee en arriere plan
        input[strlen(input) - 1] = 0; //on enleve le & de la commande
    }

    //executer les commandes internes (cd, pwd, history)
    if (strncmp(input, "cd", 2) == 0 || strncmp(input, "pwd", 3) == 0 || strncmp(input,
    "history", 7) == 0 || strncmp(input, "echo", 4) == 0) {
        execute_interne(input);
    } else {
        //executer les commandes externes
        pid_t pid = fork();
        if (pid == 0) {
            execute_externe(input);
            exit(0);
        } else if (pid > 0) {
            if (!bg) {
                waitpid(pid, NULL, 0);
                //attendre la fin du processus enfant si la commande n'est pas en
                arriere-plan
                //sinon on continue et laisse l'enfant derriere
            }
        } else {
            //si le fork echoue
            perror("fork");
        }
    }
}

void execute_command(char* input){
    //on fait un tab de commande en decoupant si il y a des ";" pour executer plusieurs
    commandes
    char* pch = strtok(input, ";");
    while (pch != NULL) {

```

```

        //on execute chaque commande
        //on coupe les espaces avants (pose soucis pour les commandes internes qui sont
pas detecte " cd" != "cd")
        while (isspace(*pch)) pch++;
        execute_single_command(pch);
        //on passe a la commande suivante
        pch = strtok(NULL, ";");
    }
}

int main() {
    //definir la fonction de completion de commande
    rl_attempted_completion_function = command_completion;

    while (1) {
        //afficher le repertoire courant avant le prompt "mbash>"
        char cwd[MAX_CMD_LEN];
        if (getcwd(cwd, sizeof(cwd)) != NULL) {
            printf("%s ", cwd);
        }
        //lire l'entree utilisateur
        char* input = readline("mbash> ");
        if (!input) {
            break; //sortir de la boucle si l'entree est NULL (EOF)
        }

        if (strlen(input) > 0) {
            //ajouter la commande à l'historique de readline et a notre historique
            add_history(input);
            add_to_history(input);
        }

        int count = 100;
        int est_historique = 0;
        do {
            if (count-- == 0) {
                printf("erreur, trop grand ou boucle infini d'appelle a l'historique\n");
                break;
            }
            est_historique = 0;
            if (input[0] == '!') {
                //gerer les commandes de l'historique
                int index;
                if (input[1] == '-') {
                    //si c'est moins on prend le dernier -1
                    index = history_count - atoi(input + 2) - 1;
                } else {
                    //sinon on prend le nombre directement
                    index = atoi(input + 1) - 1;
                }

                //si l'index est valide on recupere la commande correspondante

```

```

        if (index >= 0 && index < history_count) {
            if (history[index][0] == '!') {
                //si la commande est une commande de l'historique
                //on met est_historique a 1 pour refaire une boucle
                est_historique = 1;
                strcpy(input, history[index]);
            } else {
                strcpy(input, history[index]);
            }
        } else {
            printf("commande introuvable dans l'historique ou d'historique\n");
            continue;
        }
    }
} while (est_historique == 1);

if (strncmp(input, "exit", 4) == 0) {
    free(input);
    break; //sortir de la boucle si la commande est "exit"
}

execute_command(input);

free(input); //liberer la memoire allouee pour l'entree utilisateur
}

//liberer la memoire allouee pour l'historique
for (int i = 0; i < history_count; i++) {
    free(history[i]);
}

return 0;
}

```


2 - Serveur de Package Debian

```
gpg --full-generate-key
gpg --export --armor "Baptiste Hennequin" > public.key
gpg --export-secret-keys --armor "Baptiste Hennequin" > private.key
mkdir -p mbash-0.1/{DEBIAN,usr/bin}
```

ajouter le script dans usr/bin/

dans DEBIAN/control mettre :

```
Package: mbash
Version: 0.1
Section: utils
Priority: optional
Architecture: all
Maintainer: Baptiste Hennequin <baptiste.hennequin270@gmail.com>
Description: Un "bash" refait tout beau tout neuf
```

faire le fichier deb :

```
dpkg-deb --build mbash-0.1
et l'ajouter dans notre dépôt.
```

créer l'arborescence du dépôt :

```
mkdir -p {conf,incoming,dists,db,pool}
dans conf créer le fichier distributions :
Codename: stable
Suite: stable
Components: main
Architectures: amd64
SignWith: baptiste.hennequin270@gmail.com
```

```
reprepro -b depot includedeb stable depot/mbash-0.1.deb
```

(le package dpkg-sig ne peut pas être installé donc j'ai utilisé gpg pour signer)
gpg --output mbash-0.1.deb.sig --detach-sign --armor mbash-0.1.deb

```
mkdir -p SAE/{conf,incoming,dists,db,pool}
```

pour le serveur apache :

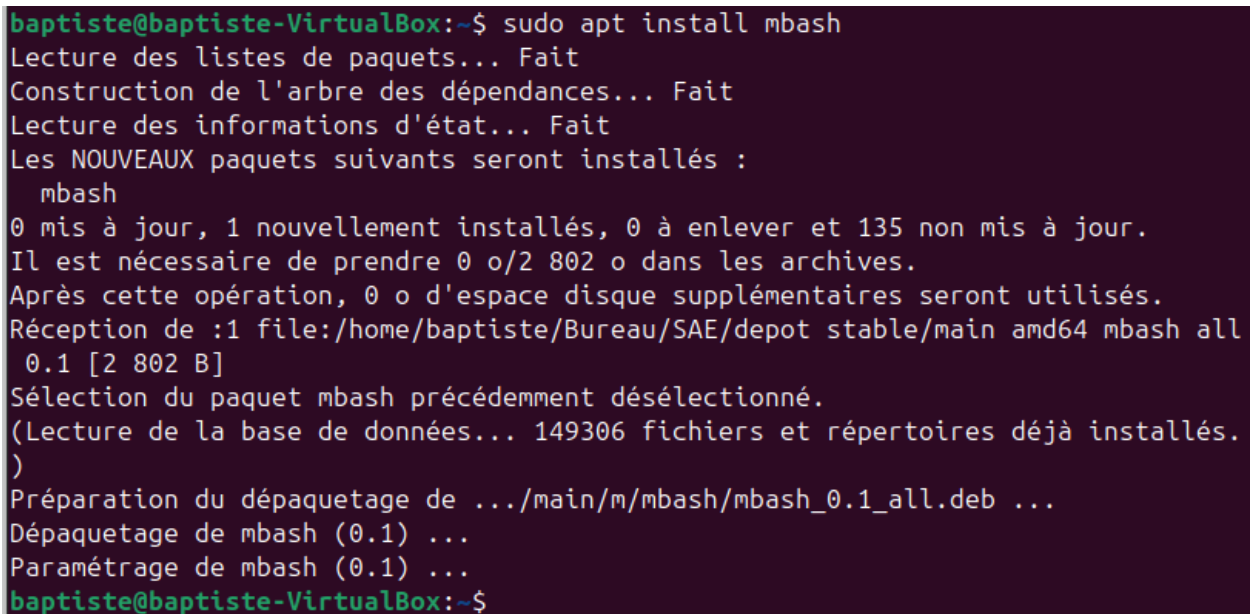
```
dans /etc/apache2/sites-available/mbash-repo.conf
<VirtualHost *:80>
    ServerName sae
    DocumentRoot /home/baptiste/Bureau/SAE/depot
    <Directory /home/baptiste/Bureau/SAE/depot>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>
</VirtualHost>
```

Pour essayer en local en local :

ajouter

deb [trusted=yes] file:/home/baptiste/Bureau/SAE/depot stable main

dans /etc/apt/sources.list



```
baptiste@baptiste-VirtualBox:~$ sudo apt install mbash
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les NOUVEAUX paquets suivants seront installés :
  mbash
0 mis à jour, 1 nouvellement installés, 0 à enlever et 135 non mis à jour.
Il est nécessaire de prendre 0 o/2 802 o dans les archives.
Après cette opération, 0 o d'espace disque supplémentaires seront utilisés.
Réception de :1 file:/home/baptiste/Bureau/SAE/depot stable/main amd64 mbash all
 0.1 [2 802 B]
Sélection du paquet mbash précédemment désélectionné.
(Lecture de la base de données... 149306 fichiers et répertoires déjà installés.
)
Préparation du dépaquetage de .../main/m/mbash/mbash_0.1_all.deb ...
Dépaquetage de mbash (0.1) ...
Paramétrage de mbash (0.1) ...
baptiste@baptiste-VirtualBox:~$
```

```
sudo ln -s /home/baptiste/Bureau/SAE/depot /var/www/html/SAE
```

pour que le serveur apache accède au dépôt.

```
sudo chmod -R o+rx /home/baptiste/Bureau/SAE
```

```
sudo chmod -R o+rx /home/baptiste/Bureau
```

```
sudo chmod -R o+rx /home/baptiste
```

pour le client :

```
sudo apt-key add /chemin/vers/public.key
```

ajouter dans /etc/apt/sources.list :

```
deb http://localhost/SAE stable main
```

Mettre à jour le package :

recréer un fichier .deb en modifiant la config dans DEBIAN/conf avec la nouvelle version.

```
baptiste@baptiste-VirtualBox:~$ apt list --installed | grep mbash

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

mbash/now 0.2 all [installé, local]
baptiste@baptiste-VirtualBox:~$ mbash
/home/baptiste mbash> ls
Bureau      Images      Musique     snap          Vidéos
Documents   Modèles     Public      Téléchargements
/home/baptiste mbash> history
1 ls
2 history
/home/baptiste mbash>
baptiste@baptiste-VirtualBox:~$
```