

---

# Term Project

---

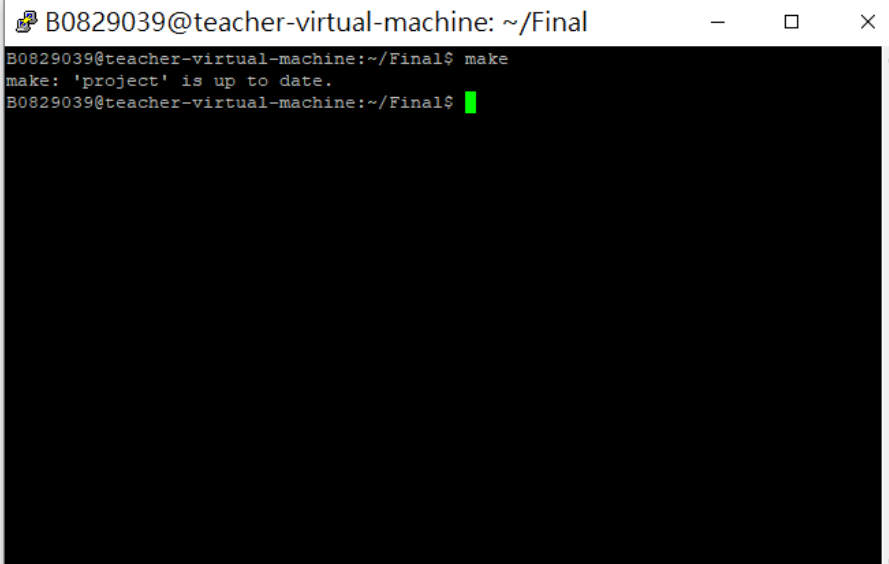
Login Shell



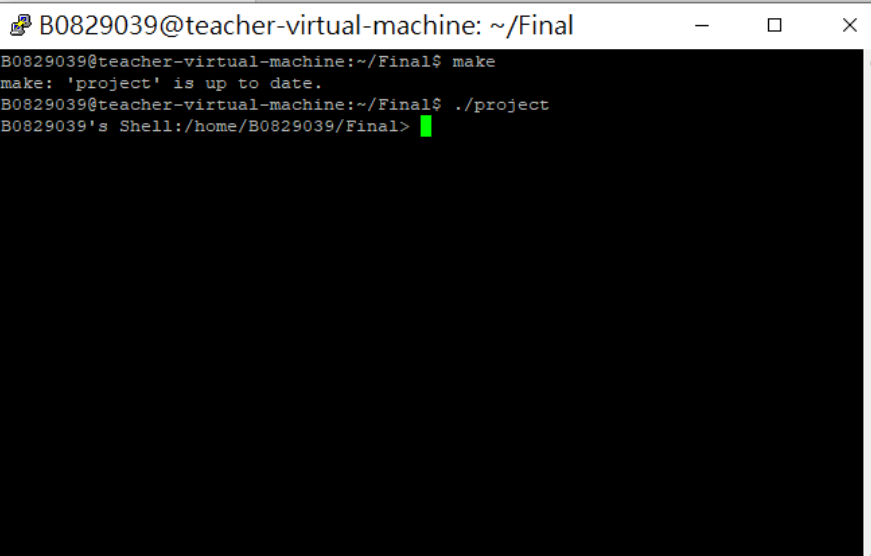
2021 年 1 月 6 日

使用說明:

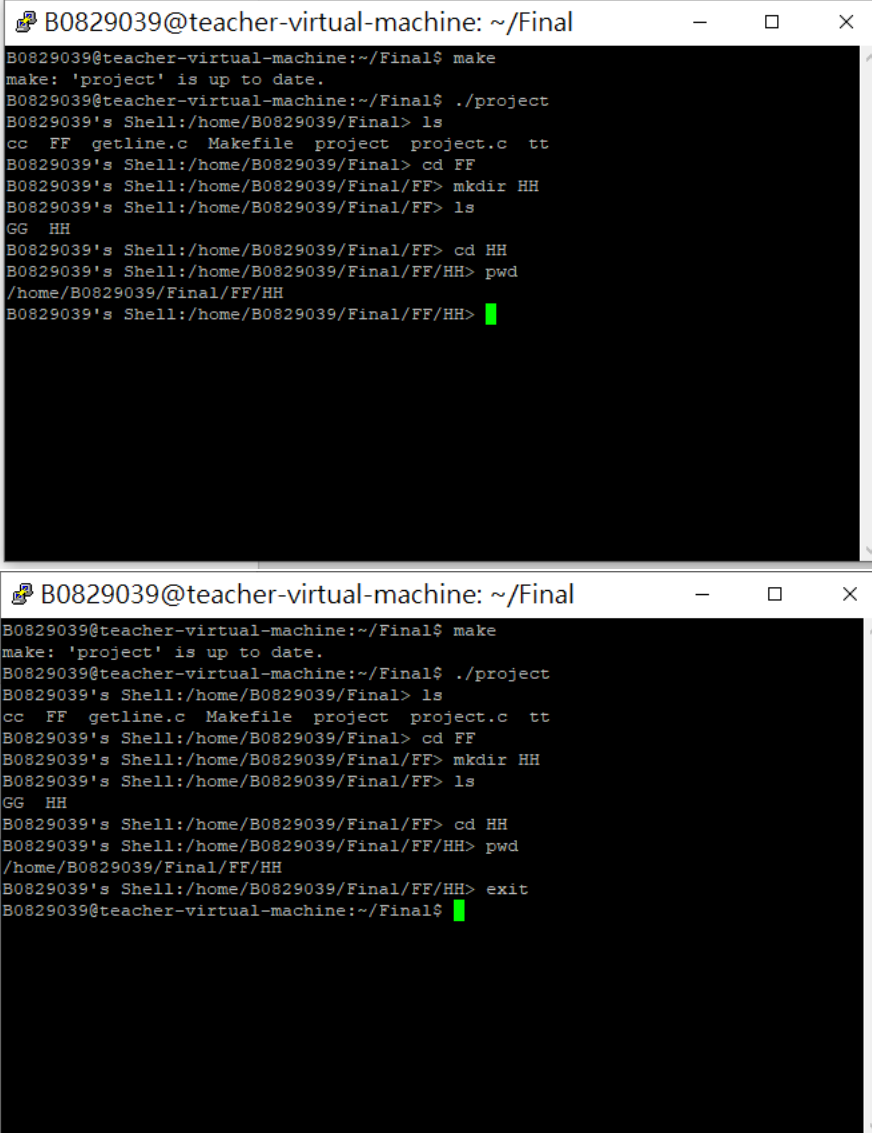
首先先將 `project.c` 與 `Makefile` 一起放在同個資料夾中，接者輸入 `make` 或是 `gcc project.c -o project` 進行編譯，然後以 `./project` 執行產生出來的檔案，便可以進入該 `shell` 並執行多種指令如 `ls,env,cd,pwd,mkdir` 等等，若要中斷執行可輸入 `exit` 或使用 `control+c`。



```
B0829039@teacher-virtual-machine: ~/Final
B0829039@teacher-virtual-machine:~/Final$ make
make: 'project' is up to date.
B0829039@teacher-virtual-machine:~/Final$
```



```
B0829039@teacher-virtual-machine: ~/Final
B0829039@teacher-virtual-machine:~/Final$ make
make: 'project' is up to date.
B0829039@teacher-virtual-machine:~/Final$ ./project
B0829039's Shell:/home/B0829039/Final>
```



```
B0829039@teacher-virtual-machine: ~/Final
B0829039@teacher-virtual-machine:~/Final$ make
make: 'project' is up to date.
B0829039@teacher-virtual-machine:~/Final$ ./project
B0829039's Shell:/home/B0829039/Final> ls
cc FF getline.c Makefile project project.c tt
B0829039's Shell:/home/B0829039/Final> cd FF
B0829039's Shell:/home/B0829039/Final/FF> mkdir HH
B0829039's Shell:/home/B0829039/Final/FF> ls
GG HH
B0829039's Shell:/home/B0829039/Final/FF> cd HH
B0829039's Shell:/home/B0829039/Final/FF/HH> pwd
/home/B0829039/Final/FF/HH
B0829039's Shell:/home/B0829039/Final/FF/HH>
B0829039@teacher-virtual-machine: ~/Final
B0829039@teacher-virtual-machine:~/Final$ make
make: 'project' is up to date.
B0829039@teacher-virtual-machine:~/Final$ ./project
B0829039's Shell:/home/B0829039/Final> ls
cc FF getline.c Makefile project project.c tt
B0829039's Shell:/home/B0829039/Final> cd FF
B0829039's Shell:/home/B0829039/Final/FF> mkdir HH
B0829039's Shell:/home/B0829039/Final/FF> ls
GG HH
B0829039's Shell:/home/B0829039/Final/FF> cd HH
B0829039's Shell:/home/B0829039/Final/FF/HH> pwd
/home/B0829039/Final/FF/HH
B0829039's Shell:/home/B0829039/Final/FF/HH> exit
B0829039@teacher-virtual-machine:~/Final$
```

程式細節解釋:

本程式通過 `shell_loop` 函數不斷地跑無窮迴圈以應對使用者的輸入，過程中會執行 `shell_read_line()` 來讀取使用者輸入的字串，以 `getline()` 函數分配記憶體並將其儲存，`shell_split_line()` 則將儲存的輸入以空格進行分割，最後以二維陣列的形式儲存指令及變數，最後再用 `shell_execute()` 執行儲存的指令，過程中會去判斷指令是否存在，是否為可直接執行的內置指令，或著是需要以 `fork` 加開進程的外置指令，並做出對應行動。

完整程式碼(含註解):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define TOKEN_BUFSIZE 64
#define MAX_DIR_NAME 2048
#define TOK_WORD " \t\r\n\a"//由於 getline 會留下一個換行字符，因此除將字串
//移除空格外也要移除換行符號
int shell_build_in_cmd(char **args){
    if(strcmp(args[0],"exit")==0)
    {
        exit(0);
    }
    else if(strcmp(args[0],"cd")==0)
    {
        if(chdir(args[1]))
        {
            printf("cd %s:no such directory\n", args[1]);
        }
        return 1;
    }//用 chdir 來改變目錄
    else if(strcmp(args[0],"pwd")==0)
    {
        char buffer[MAX_DIR_NAME]={0};
        printf("%s\n", getcwd(buffer,sizeof(buffer)));
        return 1;
    }//用 getcwd 來確認目前所在的目錄
    else
    {
        return 0; //非以上三種內置指令，需要進行 Fork 才可執行，因此 return 0
    }
}
char *shell_read_line()
{
    char *line = NULL;
```

```

    ssize_t bufsize = 0; // 利用 getline 來自動分配緩衝區，將整串輸入放入到
line 中
    getline(&line, &bufsize, stdin);
    return line;
};

char **shell_split_line(char *line){
    int bufsize = TOKEN_BUFSIZE, position = 0;
    char **commands = malloc(bufsize * sizeof(char*)); // 分配記憶體用以分
析以空格分割的輸入，先假設有 64 個區段
    char *command;
    if (!commands) {
        fprintf(stderr, "shell allocation error\n"); // 若記憶體分配失敗則輸
出錯誤資訊，並停止執行
        exit(1);
    }
    command = strtok(line, TOK_WORD); // 利用 strtok 分離出指令的第一個詞，如
cd final 中的 cd
    while (command != NULL) {
        commands[position] = command; // 在 commands[position] 儲存指令
        position++;
        if (position >= bufsize) { // 如果片段太多，超過 BUFSIZE 就繼續分配更
多記憶體給 commands
            bufsize += TOKEN_BUFSIZE;
            commands = realloc(commands, bufsize * sizeof(char*)); // 利用
realloc 在保留內容物的情況下增加記憶體空間
            if (!commands) {
                fprintf(stderr, "shell allocation error\n"); // 若 command
分配後反而為 null 則終止程式
                exit(1);
            }
        }
        command = strtok(NULL, TOK_WORD); // 繼續分割字串的剩餘部分，由於
getline 會留下一個換行字符，因此除將字串移除空格外也要移除換行符號
    }
    commands[position] = NULL; // 最後一位設成 NULL
    return commands;
};

int shell_execute(char **args)

```

```

{
    if(args[0]==NULL)//如果沒有指令則跳過這輪的執行
    {
        return 1;
    }
    else if(shell_build_in_cmd(args))//是內建指令，直接執行
    {
        return 1;
    }
    else{          //由於外置指令需要重開進程才能執行，因此 fork 後再執行
        int pid=fork();
        if(pid==0)
        {
            if(execvp(args[0],args)<0)//執行外置指令
            {
                printf("%s:command not found.\n", args[0]);//指令不存在
                exit(0);
            }
        }
        else if(pid<0)
        {
            printf("fork error\n");
            exit(1);
        }
        wait(pid);
    }
    return 1;
};

void shell_loop()
{
    char *line;
    char **args;
    char buffer[MAX_DIR_NAME];
    while (1)
    {
        printf("B0829039's Shell:%s> ", getcwd(buffer, sizeof(buffer)));
        line = shell_read_line();//接收輸入的字串
        args = shell_split_line(line);//分割輸入的字串
    }
}

```

```
        shell_execute(args);//執行指令
        free(line);//回收記憶體
        free(args);//回收記憶體
    }
};

int main()
{
    shell_loop();
    return 0;
}
```