

# Experiment Report Two

Course Name: Operating System

Experiment Name: Making a simple shell

Student Name: 李添豪

Student ID: 5140309349

Class Number: F1403015

Catalogue

[Experiment Report Two 1](#_Toc450379787)

[Experiment Environment 3](#_Toc450379788)

[Experiment Target 3](#_Toc450379789)

[Experiment Design 3](#_Toc450379790)

[Function main(): 3](#_Toc450379791)

[Accepting Command 3](#_Toc450379792)

[Run command 4](#_Toc450379793)

[Experiment Code 4](#_Toc450379794)

[Test： 9](#_Toc450379795)

[Summarize: 9](#_Toc450379796)

## Experiment Environment

Operating System: Ubuntu LTS 16.04 (based on VMware)

Kernel Version: 4.4.6

## Experiment Target

1. Make a simple shell that can accept bash command.
2. Provide a background choose to execute the command (when command ends with ‘&’)
3. Provide a history feature.
   1. When program accepts a Ctrl-C interrupt, print the last 10 commands.
   2. When user input the “r x” command, run the command begin with character ‘x’ when there are several commands beginning with ‘x’ run the latest one.
   3. When user input the ‘r’ command, run the last command.
   4. “r” or “r x” command should not appear in the history command but the command to run by after run “r” or “r x”

\*Bonus:

In this project, realize the cd build-in command in BASH via system call chdir() system call. And for other build-in command that cannot realize via execvp(), it’s just some basic works like the fulfillment of cd.

## Experiment Design

### Function main():

1. Change current path into home
2. Register the interrupt handle for Ctrl-C use function signal()
3. Loop:
   1. Print the header like: emile@ubuntu:~$
   2. Accepting Command
4. Return

### Accepting Command

1. Use a char\* list to accept command.
2. if (command is a r or r x command):

choose one history command, run it and make a record in history list.

else

run the command and make a record of it in history list.

### Run command

if（command is ‘cd’）fulfill it via chdir();

else fulfill it via execvp();

## Experiment Code

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

#include<malloc.h>

#include<signal.h>

#include<string.h>

#define NONE "\033[m"

#define LIGHT\_BLUE "\033[1;34m"

#define COLOR\_LIGHT\_CYAN "\033[1;36m"

#define COLOR\_PURPLE "\033[0;35m"

#define MAX\_ARGS 20

char **\*\***command**;**

char **\*\***commandList**[**1000**]={**0**};**

int commandNow**=**0**;**

void acceptCommand**();**

void execute**(**int commandNum**);**

void handle**(**int signalNum**);**

int main**()**

**{**

chdir**(**"/home/emile"**);**

char **\***file\_path\_getcwd**;**

file\_path\_getcwd**=(**char **\*)**malloc**(**60**);**

signal**(**SIGINT**,**handle**);** //Set handle of SIGINT signal

**while(**1**){**

//ask for a space for current command

command **=(**char **\*\*)** calloc**(**MAX\_ARGS**,sizeof(**char**\*));**

//print the header

getcwd**(**file\_path\_getcwd**,**60**);**

printf**(**LIGHT\_BLUE"emile@lsh"NONE**);**

printf**(**":"**);**

printf**(**COLOR\_LIGHT\_CYAN"%s"NONE**,!(**strcmp**(**file\_path\_getcwd**,**"/home/emile"**))?**"~"**:**file\_path\_getcwd**);**

printf**(**"$ "**);**

fflush**(**stdout**);**

acceptCommand**();**

**}**

**}**

void acceptCommand**()**

**{**

int i**=**0**;**

**while(**1**){**

command**[**i**]=(**char **\*)**calloc**(**20**,sizeof(**char**));**

scanf**(**"%s"**,**command**[**i**++]);**

char signal**=**getchar**();**

**if(**i**==**80**)break;**

**if(**signal**==**'\n'**)break;**

**}**

**if(!**strcmp**(**command**[**0**],**"r"**))**

**{**

//for r and r x command

**if(**command**[**1**]==NULL)**

**{**

//r

commandList**[**commandNow**]=**commandList**[**commandNow**-**1**];**

**++**commandNow**;**

execute**(**commandNow**-**1**);**

**}**

**else{**

//r x

char x**=\***command**[**1**];**

**for(**int i**=**commandNow**-**1**;**i**>=**0**&&**i**>=**commandNow**-**10**;--**i**){**

**if(\***commandList**[**i**][**0**]==**x**)** **{**

commandList**[**commandNow**++]=**commandList**[**i**];**

execute**(**i**);**

**return;**

**}**

**}**

printf**(**"No Command found!\n"**);**

fflush**(**stdout**);**

**}**

**}**

**else{**

commandList**[**commandNow**++]=**command**;**

execute**(**commandNow**-**1**);**

**}**

**return** **;**

**}**

void execute**(**int commandNum**)**

**{**

//Copy the target command for modifying.

char **\*\***command\_p**=**commandList**[**commandNum**];**

char **\*\***temp**;**

temp**=(**char **\*\*)** calloc**(**MAX\_ARGS**,sizeof(**char**\*));**

**for(**int i**=**0**;**i**<**MAX\_ARGS**;++**i**){**

**if(**command\_p**[**i**]){**

temp**[**i**]=(**char **\*)**calloc**(**20**,sizeof(**char**));**

strcpy**(**temp**[**i**],**command\_p**[**i**]);**

**}**

**else** **break;**

**}**

**if(!**strcmp**(**temp**[**0**],**"cd"**)){**

**if(**temp**[**1**][**0**]==**'~'**)**

**{**

int error**=**0**;**

char **\*** path**;**

path**=(**char **\*)** calloc**(**80**,sizeof(**char**));**

strcpy**(**path**,**"/home/emile"**);**

int i**=**1**;**

**while(**temp**[**1**][**i**])**

**{**

path**[**10**+**i**]=**temp**[**1**][**i**];**

**++**i**;**

**}**

path**[**10**+**i**]=**0**;**

error**=**chdir**(**path**);**

**if(**error**==-**1**)**

**{**

printf**(**"emile@lsh: cd: %s: No such file or directory\n"**,**path**);**

**}**

**}**

**else**

**{**

int error**;**

error**=**chdir**(**temp**[**1**]);**

**if(**error**==-**1**)**

**{**

printf**(**"emile@lsh: cd: %s: No such file or directory\n"**,**temp**[**1**]);**

**}**

**}**

**}**

**else{**

//find the last argue position;

int bgflag**=**0**;**

**while(**temp**[**bgflag**])** **++**bgflag**;**

**--**bgflag**;**

//call temp command

pid\_t pid**;**

**if(\***temp**[**bgflag**]==**'&'**){** //Background Mode

temp**[**bgflag**]=NULL;**

pid**=**fork**();**

**if(**pid**==**0**)** execvp**(**temp**[**0**],(**char **\*** const**)**temp**);**

**}**

**else{** //Front Mode

int i**;**

pid**=**fork**();**

**if(**pid**==**0**)** **{**

i**=**execvp**(**temp**[**0**],(**char **\*** const**)**temp**);**

**if(**i**==-**1**)** exit**(NULL);**

**}**

**else** waitpid**(**pid**,NULL,**0**);**

**}**

**}**

free**(**temp**);**

**return** **;**

**}**

void handle**(**int signalNum**)**

**{**

**if(**signalNum**==**2**){**

//Set the begin command to print

int pos**=**commandNow**>=**10**?**commandNow**-**10**:**0**;**

**for(**int i**=**pos**;**i**<**commandNow**;++**i**)**

**{**

printf**(**"\n%d: "**,**i**);**

char **\*\***arg**=**commandList**[**i**];**

**while** **(\***arg**)** printf**(**"%s "**,\*(**arg**++));**

fflush**(**stdout**);**

**}**

//print the input hint;

printf**(**LIGHT\_BLUE"\nemile@lsh:~$ "NONE**);**

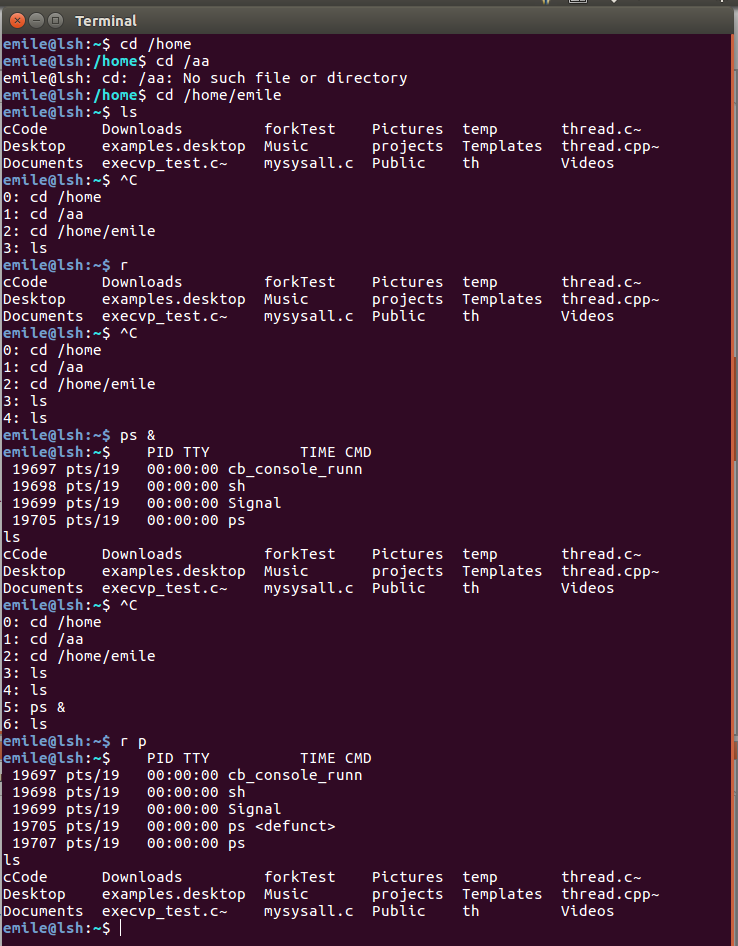
fflush**(**stdout**);**

**}**

**return;**

**}**

## Test：



## Summarize:

Log:

Version 1.001

Fixing one bug: when input a wrong command then send a Ctrl-C interrupt, history command will be printed twice.

Adding a built-in command of bash: cd.

In this project, function execvp() is the most important, learning how to use it proper is the key point of the experiment. For example, the parameters list you want to pass to the command through execvp(), the last element should set as NULL.

The logical part of this program is also a small challenge, and the pointer is really annoying, you need to handle it carefully. An overall design before you set about programming is also essential.

Another hint is some command you can run in BASH, you cannot realize it using execvp(), because some command like cd is a build-in command which means it is accomplished in the code of bash. What you can do to accomplish such command in your shell is to establish a hash table as an index for those specially appointed command to call their own subroutine to fulfill their function.