THE ISLAMIC UNIVERSITY OF GAZA

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Operating System Lab

Developing A Shell

By:

Yasmin Abdelaal	**220201094**
Mona Asad Alhasani	**220201093**
Hala Abu Zerieq	**220223352**

Submitted for

Eng. AMAL MAHFOUTH

Project Overview

The main objective of this project is to build a basic shell (called myshell) that mimics the core functionality of traditional Unix/Linux shells. It provides built-in support for commands like cd, pwd, echo, dir, environ, help, pause, and quit. It also supports advanced features such as input/output redirection, background execution using & symbol, and batch mode execution via .bat files.

Implemented Features

- Internal Commands: cd,clr, pwd, echo, dir, environ, help, pause, quit

- Input Redirection using `<`

- Output Redirection using `>` and `>>`

- Background execution using `&`

- Batch file support using `.bat`

Execution Flow

The shell starts by displaying a welcome message and waits for user input. The input is parsed into tokens and redirected or executed based on the command. The shell supports both interactive and non-interactive (batch) modes. Help documentation is loaded from a `readme` file for command guidance.

1. myshell.c – Main Shell Implementation

This file contains the main() function that starts the shell. It handles environment variable setup, batch mode, user prompt, and calls Command_shell().

2. utility.c – Command Execution Utilities

Contains the command handlers and utilities like Command_exec(), Command_cd(), Command_dir(), Command_echo(), and redirection support.

3. myshell.h – Header File

Declares functions and includes struct Redirect and constants like MAX_BUFFER and MAX_ARGS, making the code modular.

4. makefile – Build Automation

Compiles myshell using GCC with a single make command. It includes all source and header files in the compilation process.

5. readme – Help Manual

Used by Command_help() to display command usage, syntax, and shell-related information. Accessible using help commands inside myshell.

PROJECT C SOURCE CODE:

code of MyShell.c :

```c
#include "myshell.h"

        /* Colors */
#define ANSI_COLOR_BRIGHT_RED "\033[1m\033[31m"
#define ANSI_COLOR_BIRGHT_BLUE "\033[1m\033[34m"
#define ANSI_COLOR_CYAN_BOLD  "\033[1m\033[36m"
#define ANSI_COLOR_YELLOW_BOLD   "\033[1m\033[33m"
#define ANSI_COLOR_RESET  "\033[0m"


int main (int argc, char *argv[])
{

        char buf[MAX_BUFFER], pwd[MAX_ARGS];    // line buffer
        char shell_path[MAX_ARGS]="shell=";
        char readme_path[MAX_ARGS]="readme_path=";
        char newpath[MAX_ARGS*1000];
        int len;

        strcpy(newpath,getenv("PATH"));
        strcat(newpath,":");
        if(strcmp(argv[0],"./myshell")&&strcmp(argv[0],"myshell"))
        {
                len=strlen(argv[0]);
                while(len&&argv[0][len]!='/')
                 len-- ;
                argv[0][len]='\0';
                strcpy(pwd,argv[0]);
                get_fullpath(pwd, argv[0]);
                printf("%s\n",pwd);
        }
        else
                strcpy(pwd,getenv("PWD"));
```

```c
        }
        else
                strcpy(pwd,getenv("PWD"));

        strcat(newpath,pwd);   // strcat(newpath,getenv("PWD"));
        setenv("PATH",newpath,1);// add the current working directory  in the "PATH" environment variable to search for the filename specified.
        strcat(shell_path,pwd);   //  strcat(shell_path, getenv("PWD"));
        strcat(shell_path,"/myshell");
        putenv(shell_path); //add the working directory  of myshell in the environment variables
        strcat(readme_path, pwd);
        strcat(readme_path,"/readme");
        putenv(readme_path); //   add the filepath of the file "readme"  in the environment variables, see function  my_help( ) !

        if(argc>1)  // User input directly from the terminal  ./myshell a.bat >c.txt
        {
                strcpy(buf,"myshell ");
                int i;
                for(i=1;i<argc;i++)
                {
                        strcat(buf,argv[i]);
                        strcat(buf,"  ");
                }
                Execute(buf);// execute this command(bat)£¬
        }

        else  // if user input ./myshell
        {

                Command_clear( );
                //print team members names
```

```c
                for(i=1;i<argc;i++)
                {
                        strcat(buf,argv[i]);
                        strcat(buf,"  ");
                }
                Execute(buf);// execute this command(bat)£¬
        }

        else  // if user input ./myshell
        {

                Command_clear( );
                //print team members names
        printf("\033[1;32m");
        printf("Team Members:\n");
        printf("-Yasmin Abdelaal\n");
        printf("-Mona Alhasani\n");
        printf("-Hala Abu Zerieq\n");
        printf("\033[0m\n");
                fprintf(stderr, ANSI_COLOR_CYAN_BOLD "Welcome to this simple shell!\n" ANSI_COLOR_RESET);

                printf("Type ");
                printf(ANSI_COLOR_BIRGHT_BLUE "help" ANSI_COLOR_RESET);
                printf(" to view manual\n");

                printf("Type ");
                printf(ANSI_COLOR_BRIGHT_RED "exit" ANSI_COLOR_RESET);
                printf(" to terminate.\n");

                Command_shell(stdin,NULL,NULL);
        }
    return 0 ;
}    // end  function "main"
```

code of myshell.h

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <errno.h>
#include <dirent.h>
#include <ctype.h>


#define MAX_BUFFER 1024  // max line buffer
#define MAX_ARGS 64  // max # args
#define SEPARATORS " \t\n"  // token sparators
#define NUM 10        // the command number
#define MAX_OPEN 10  //  open 10 stdin redirection files max and 10 stdout redirection files, myshell support several I/O redirection£©
#define MAX_PATH 100 //  the maxium length of file and forder



/*********************************************************************
announce certain data types
*********************************************************************/
typedef struct// redirection date structure
{
        char *filename;   // redirection file name
        char opentype[3]; // the open method of redirection files "a"   "r"   "w"
        char open[3];  // the open method of redirection files  ">>"   "<"   ">"
} Redirect;          //

extern int errno;        // system error number
extern char **environ;   // environment array
```

```c
extern int errno;        // system error number
extern char **environ;   // environment array


/*********************************************************************
announce the functions
*********************************************************************/

int Execute(char *buffer);//the execution of the command
int Command_exec(char **args, const Redirect *Inputs,const Redirect *Outputs,int *states);//
int Error(int errortype,char **args,const Redirect *  IOputs,const int *states, char * msg) ;//error message printout
int Command_strtok(char *buf,char **args,int *states,Redirect *InPuts,Redirect *OutPuts);//analyse the command
int Command_cd(char **args,const Redirect *Inputs, int *states);// execute command 'cd'
void Command_clear(void);// execute command 'clear'
int Command_dir(char **args,const Redirect *Inputs, int *states);// execute command 'dir'
int Command_echo(char **args,const Redirect *Inputs,int *states);// execute command 'echo'
int list_environ(void);// execute the command "environ"
int show_pwd(void); // execute command 'pwd'
int Command_shell(FILE *inputfile,const Redirect *Outputs,const int *states);//keep reading a line from stdin or inputfile and call "Execute()" to execute the command line.
void Command_delay(int n);           //delay to ensure the order of processes
void get_fullpath(char *fullpath,const char *shortpath);  //get the full path of a file or a directory
int Command_help(char **args,const Redirect *Outputs,int *states);// display the user manual;    seek  for  the key word such as  <help  dir> , output the file "readme" until meeting  '#'
```


```c
extern char **environ;   // environment array


/*********************************************************************
announce the functions
*********************************************************************/

int Execute(char *buffer);//the execution of the command
int Command_exec(char **args, const Redirect *Inputs,const Redirect *Outputs,int *states);//
int Error(int errortype,char **args,const Redirect *  IOputs,const int *states, char * msg) ;//error message printout
int Command_strtok(char *buf,char **args,int *states,Redirect *InPuts,Redirect *OutPuts);//analyse the command
int Command_cd(char **args,const Redirect *Inputs, int *states);// execute command 'cd'
void Command_clear(void);// execute command 'clear'
int Command_dir(char **args,const Redirect *Inputs, int *states);// execute command 'dir'
int Command_echo(char **args,const Redirect *Inputs,int *states);// execute command 'echo'
int list_environ(void);// execute the command "environ"
int show_pwd(void); // execute command 'pwd'
int Command_shell(FILE *inputfile,const Redirect *Outputs,const int *states);//keep reading a line from stdin or inputfile and call "Execute()" to execute the command line.
void Command_delay(int n);           //delay to ensure the order of processes
void get_fullpath(char *fullpath,const char *shortpath);  //get the full path of a file or a directory
int Command_help(char **args,const Redirect *Outputs,int *states);// display the user manual;    seek  for  the key word such as  <help  dir> , output the file "readme" until meeting  '#'
```


```c
int Command_bat(char **args,const Redirect *Inputs,const Redirect *Outputs,int *states); // execute the command "myshell" with a batchfile
```

code of utility.c:

Note:

The code is too long, so we included several screenshots to show that we executed it. You can also see the details in the files we submitted.

```c
#include "myshell.h"

/* Colors */
#define ANSI_COLOR_BRIGHT_RED "\033[1m\033[31m"
#define ANSI_COLOR_BIRGHT_BLUE "\033[1m\033[34m"
#define ANSI_COLOR_CYAN_BOLD "\033[1m\033[36m"
#define ANSI_COLOR_YELLOW_BOLD "\033[1m\033[33m"
#define ANSI_COLOR_RESET "\033[0m"
// resets the text color to the default color for the terminal.

int back_bat = 0;                  // indicate whether a process is both a background process and a batch process. 1=back , 0=not, it means it is both background process and batch, effective value of 1
int output_num = 0;                // used to store the number of output batch numbers in redirection., the number of output batch number in redirection
char batchfile[MAX_PATH]; // used to store the name of the current batch file.,
int bat_num = 0;                   // used to store the number of batch command-lines,
int isBatch = 0;                   // used to indicate whether the current process is a batch process 1 = yes , 0 = not
int letter;                                // store a single letter.
char *open;                                // store the name of a file that is to be opened

/*  Function "Execute" : used as the function system( ), it interpret the input, and call function "Command_exec" to  execute the all the commands.  */
int Execute(char *buffer) // buffer contain a command to be executed
{
        pid_t pid;
        char *args[MAX_ARGS]; /// pointers to arg strings
        int error;
        int states[5]; // states[0] is back exec; states[1] is inputfile num; states[2] is outputfile num ;
```

```c
        int states[5]; // states[0] is back exec; states[1] is inputfile num; states[2] is outputfile num ;
        // states[3] is priority of inputfile(not args); states[4] is argc ;
        Redirect Inputs[MAX_OPEN];      //   input    redirection (10) max files can be oppened at once
        Redirect Outputs[MAX_OPEN]; //   output   redirection   (10)

        error = Command_strtok(buffer, args, states, Inputs, Outputs);
        /*
                                                        تقسم النص حسب التوكنايز
        tokenizing the command in buffer into individual arguments,
        and storing the resulting array of strings in args.
        It also populates the states array and the Inputs and Outputs arrays
        with information about the command.
         :tokenize  input last  entry  will  be  NULL
         */
        if (error || args[0] == NULL)
                return -1; // If the input format error or  if there's anything there

        if (!strcmp(args[0], "quit") || !strcmp(args[0], "exit")) // if entered "quit" command
        {
                //It checks if there is an argument after the quit command.
                if (args[1])                                                            //   no argument is needed after "quit"
                        Error(-2, args + 1, NULL, NULL, args[0]);

                if (output_num > 1) // e.g.  myshell test.bat >m.txt >n.txt
                {
                        fprintf(stderr, "Exit\n");
                        return 1;
```

```c
                        return 1;
                }

                if (isBatch)
                        fprintf(stderr, "Batch file \"%s\" is finished!\n", batchfile);
                else
                        fprintf(stderr, "\n\t\tGoodbye! \n\n");
                exit(0); // break out of 'while (!feof(stdin))' loop in "main" function
        }

        else if (states[0]) // states[0]==1  running in background "flag indicate whether executed in background or not"
        {
                switch (pid = fork())
                {
                case -1:
                        Error(-9, NULL, NULL, states, "fork");//print error message and exit prog
                case 0: // child   //sleep(1);
                        Command_delay(12);
                        fprintf(stderr, "\n");
                        Command_exec(args, Inputs, Outputs, states);
                        exit(1);
                default:
                        if (isBatch == 0)
                                fprintf(stderr, "pid=%d\n", pid);
                } // end switch
        }

        else // states[0]==0  running in front
                Command_exec(args, Inputs, Outputs, states);
```

```c

        return 0;
}

/*  Function "Command_exec"  : execute the command   */
int Command_exec(char **args, const Redirect *Inputs, const Redirect *Outputs, int *states)
{
        char filepath[MAX_PATH], parent[MAX_ARGS];
        FILE *outputfile = NULL, *inputfile;
        pid_t newpid;
        int flag;

        if (!strcmp(args[0], "myshell") || !strcmp(args[0], "shell")) // strcmp() cmp function "myshell" command, checks if the first argument s either "myshell" or "shell"
        {
                flag = 0;
                if (isBatch) //     e.g.   execute myshell b.txt in test.bat
                {
                        switch (newpid = fork())
                        {
                        case -1:
                                Error(-9, NULL, NULL, states, "fork");
                        case 0:
                        // is both a background process and a batch process.
                                if (states[0] && (args[1] || states[1]))
                                {
                                        back_bat++;
                                        flag = 1;
                                }
                                output_num = states[2];
                                Command_bat(args, Inputs, Outputs, states);
```

```c
                                Command_bat(args, Inputs, Outputs, states);
                                if (flag)
                                        back_bat--;
                                output_num = 0;
                                exit(0);
                        default:
                                waitpid(newpid, NULL, WUNTRACED);
                        }
                }
                else
                {
                        if (states[0] && (args[1] || states[1]))
                        {
                                back_bat++;
                                flag = 1;
                        }
                        output_num = states[2];
                        Command_bat(args, Inputs, Outputs, states);
                        if (flag)
                                back_bat--;
                        output_num = 0;
                }
                if (states[0])
                        exit(1);
                else
                        return 0;
        }

        if (states[2]) // set output Redirection : use freopen()
        {
                get_fullpath(filepath, Outputs->filename);
                outputfile = freopen(filepath, Outputs->opentype, stdout);
                if (outputfile == NULL)
```

```c
                if (outputfile == NULL)
                {
                        Error(-6, NULL, NULL, NULL, Outputs->filename);
                        if (states[0])
                                exit(1);
                        else
                                return -4;
                }
        }

        // check for internal/external command
        if (!strcmp(args[0], "cd")) // "cd "  command
                Command_cd(args, Inputs, states);

        else if (strcmp(args[0], "clr") == 0 || strcmp(args[0], "clear") == 0) // "clear"  command

{
system("clear");
                if (output_num == 0)
                        Command_clear();                                // In Command_clear() , execute clear
                if (args[1] || states[1] || states[2]) //  no argument is needed after "clear"
                        Error(4, NULL, NULL, NULL, args[0]);
        }

        else if (!strcmp(args[0], "dir")) // "dir"  command
                Command_dir(args, Inputs, states);

        else if (!strcmp(args[0], "echo"))              // "echo" command
                Command_echo(args, Inputs, states); //

```

makefile

Activities Text Editor Jun 17 16:39

Open makefile ~/myshell_project Save

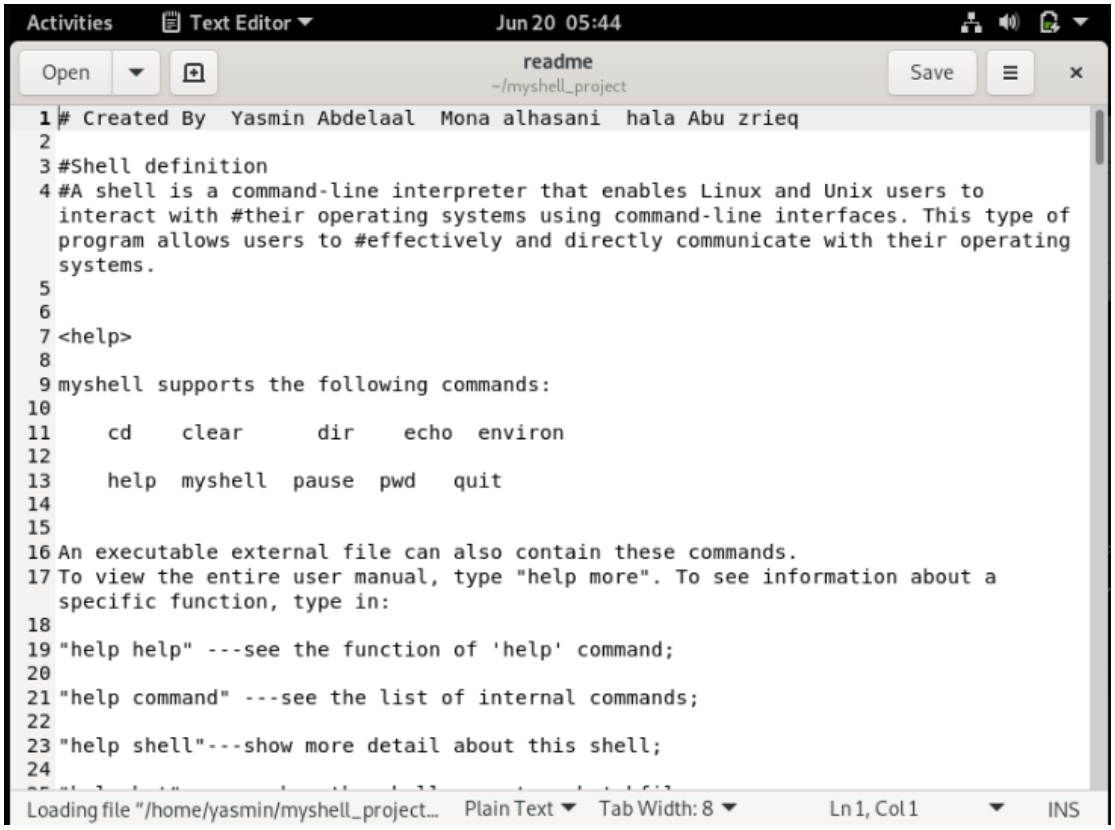
```
myshell: myshell.c utility.c myshell.h
        gcc -Wall myshell.c utility.c -o myshell
```

Makefile Tab Width: 8 Ln 1, Col 1 INS

Readem

Note

The file content is too long, so we included several screenshots to show that we executed it. You can also see the details in the files we submitted.



Activities Text Editor Jun 20 05:45

Open readme ~/myshell_project Save

```
"help bat"---see  how the shell execute a batchfile;

"help i/o redirection"---learn about the i/o redirection;

"help background"---learn about the background mode;

"help path"---see the format of filepath or dir path.

#




<help shell>

To view more details about this shell;

The shell begins by taking input from the user through standard input, which is a
file descriptor that represents anything entered on the command line. After
displaying the prompt, the shell receives the input as a string, allowing it to
move on to the next steps in the process.
The shell process involves the following steps:

   * interprets the input,

   * takes appropriate action, and
```

Plain Text Tab Width: 8 Ln 2, Col 1 INS

Activities Text Editor Jun 20 05:45

Open readme ~/myshell_project Save

```
   * takes appropriate action, and

   * finally prompts for more input.
#

<help command> <help commands> <help internal command> <help internal commands>
To view a list of internal commands:
To access detailed usage information for each command, write "help [command]"
Example:  help cd

myshell supports the following commands:

    cd    clr      dir    echo  environ

    help  myshell  pause  pwd   quit

#



<help bat> <help batchfile>


myshell is capable of obtaining its command line input from a batch file.
If myshell is invoked with a command line argument:

Example:   myshell test.bat

the batch file "test.bat" is assumed to contain a series of commands for the shell
```

Plain Text Tab Width: 8 Ln 2, Col 1 INS

Activities Text Editor Jun 20 05:45

Open readme ~/myshell_project Save

```
through the display. For Additional information about myshell you can accessed by
typing "help myshell".

#

<help i/o redirection> <help I/O Redirection> <help io redirection>

myshell supports redirecting both standard input (stdin) and standard output
(stdout) through the use of input/output (I/O) redirection.

Example:   progName arg1 arg2 <inputFile > outputFile

This command will run the program " progName " with the arguments "arg1" and
"arg2".
In this case, the standard input (stdin) file stream is replaced by " inputFile "
and the standard output (stdout) file stream is replaced by " outputFile ".
The file can be specified using the full path, such as "/home/username/test.txt".
To learn more about the file path format, type "help path" for additional
information.
Input redirection >>> stdin redirection, is supported for the internal command's
"cd", "dir", and "echo".
Output redirection >>> stdout redirection, is available for the internal command's
"dir", "environ", "echo", "help", and "pwd".


The file can be given by the full path as:

    /home/username/test.txt
```

Plain Text Tab Width: 8 Ln 2, Col 1 INS

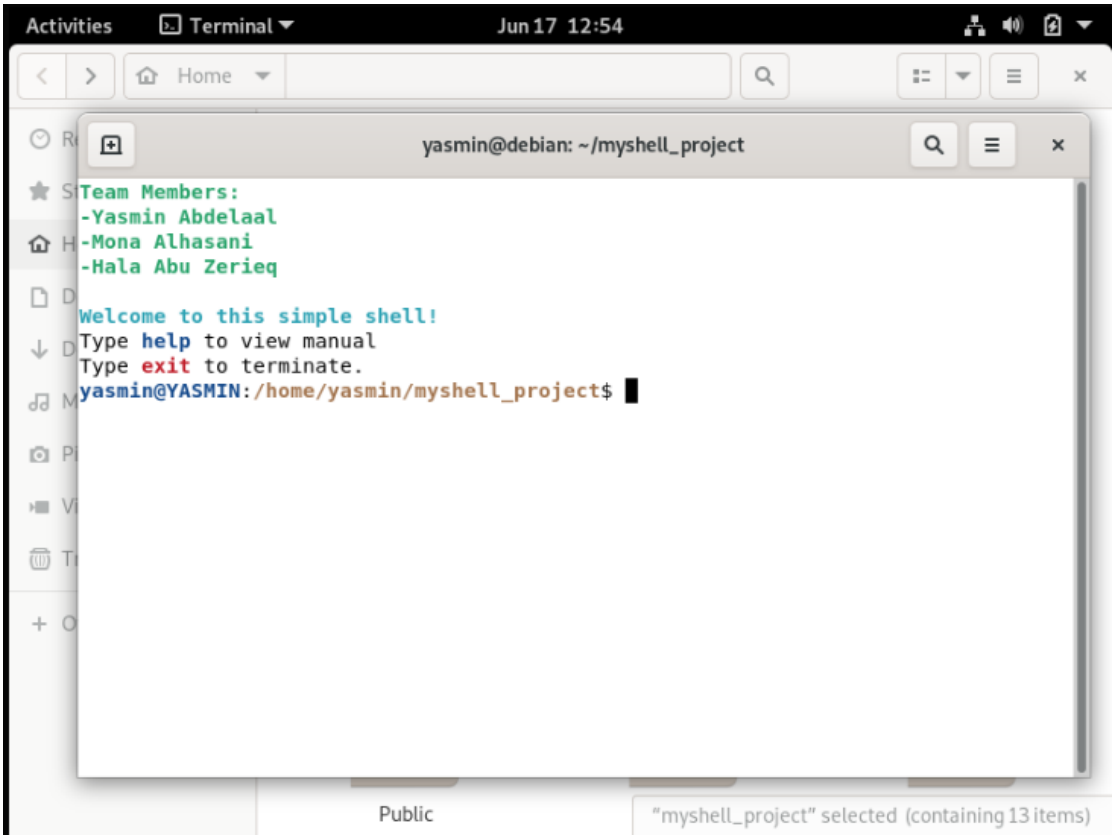
PROJECT OUTPUT:

Running the Custom Shell Successfully

Before executing any commands, we successfully launched our custom shell, myshell, as shown in Screenshot

```
yasmin@debian:~/myshell_project$ make
gcc -Wall myshell.c utility.c -o myshell
yasmin@debian:~/myshell_project$ ls
makefile  myshell  myshell.c  myshell.h  utility.c
yasmin@debian:~/myshell_project$
```

"4.png" selected (103.0 kB)



Testing Core Shell Commands

To ensure core functionality of the shell, we tested several built-in commands and verified their behavior.

cd and clear Commands:

cd Command: Used to change directory. Successfully navigates to the target path.

clear Command: Clears the shell screen. Helps in maintaining clean output during testing.

pwd Command

Purpose: Display current working directory

Result: Displayed the correct directory, confirming internal location tracking.

dir Command

Purpose: List files and folders in the current directory

Result: Displayed directory contents as expected.

echo Command

Purpose: Display a message

Result: Output matched input, confirming basic I/O functionality. Command used: echo free Palestine

environ Command

Purpose: Display environment variables

Result: Listed variables like PATH, PWD, confirming environment setup.

pause Command

Purpose: Pause shell until Enter is pressed

Result: Shell paused successfully and resumed after user input.

help Command

Purpose: Display help info from README

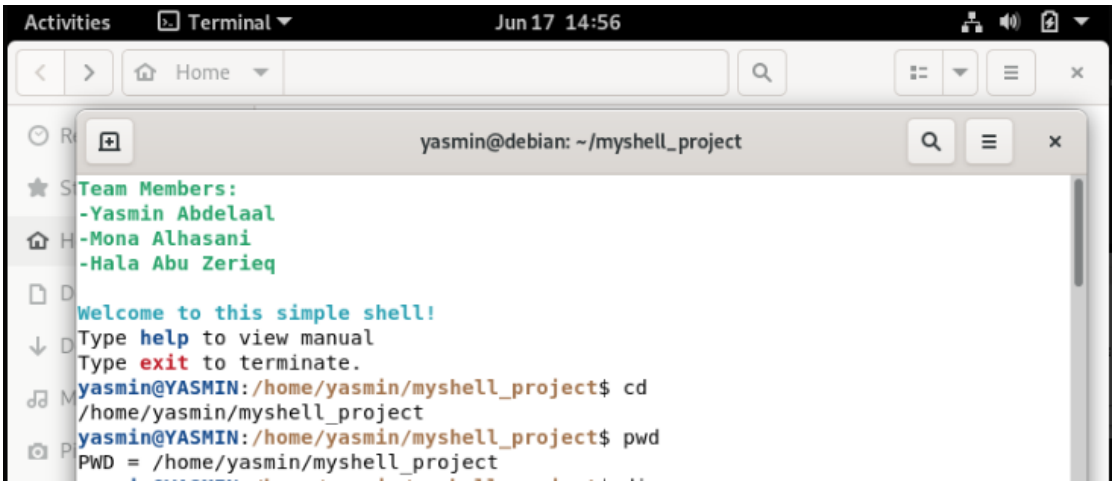
Result: Successfully read help content from file and displayed it, confirming string matching and file reading logic.

quit Command

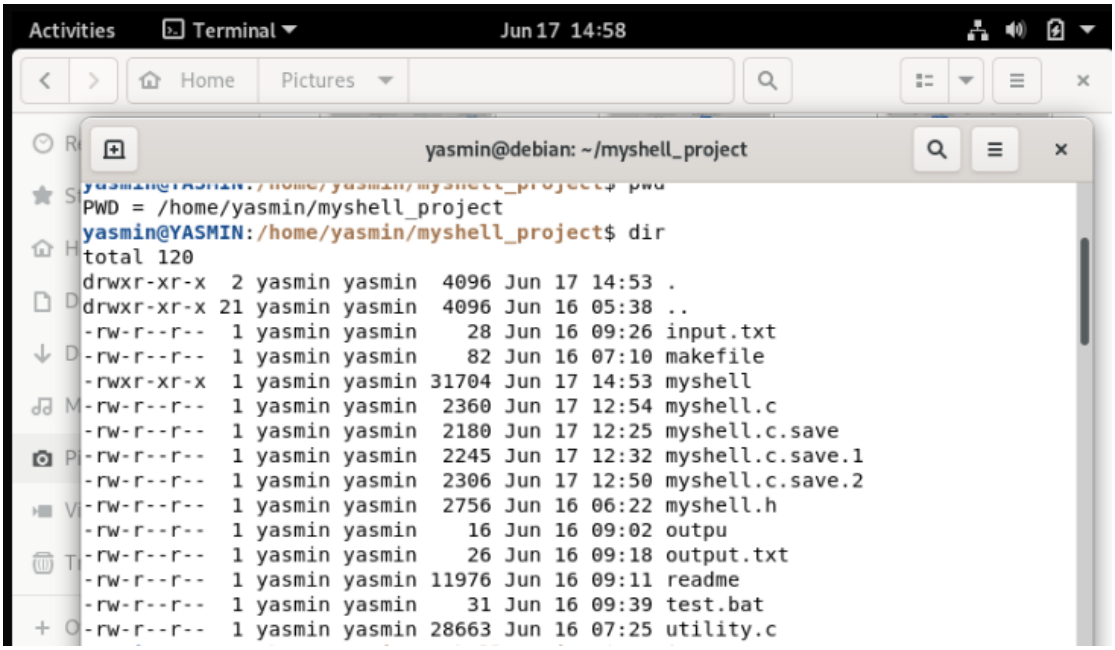
Purpose: Exit shell

Result: Displayed exit message and returned control to terminal, proving graceful termination.

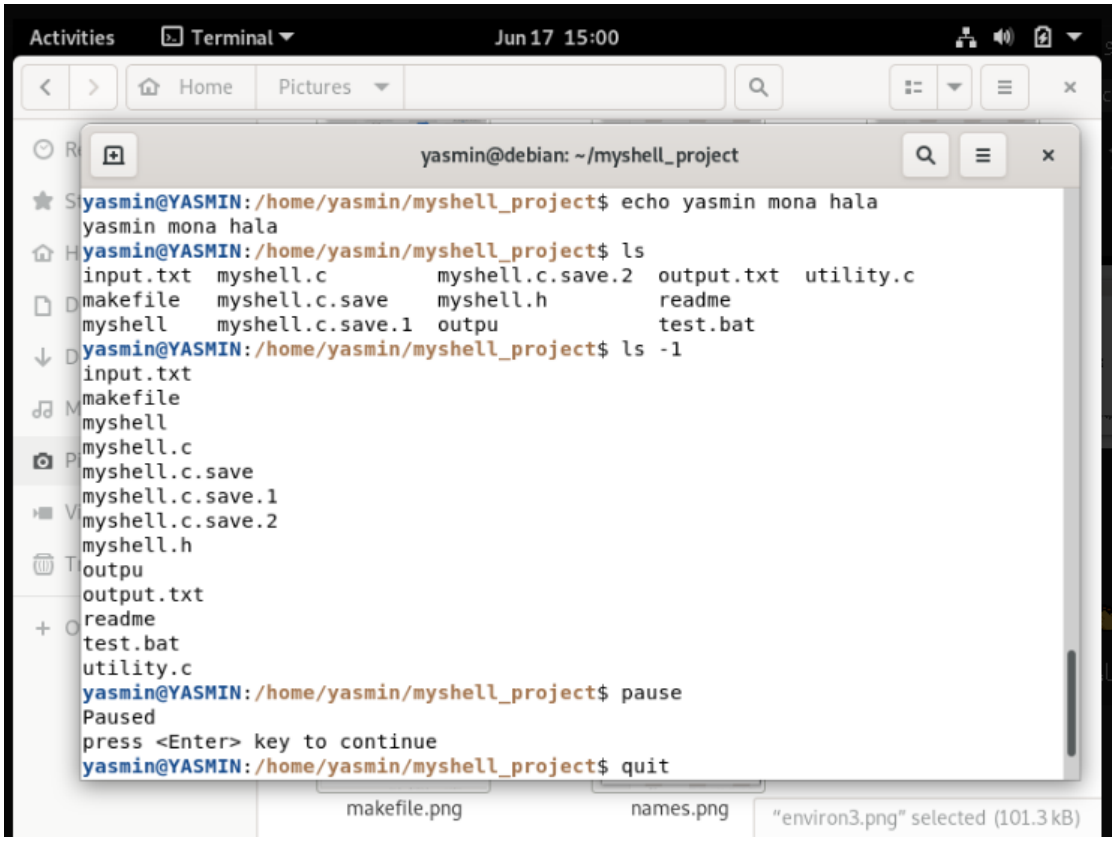
Cd , pwd commands:



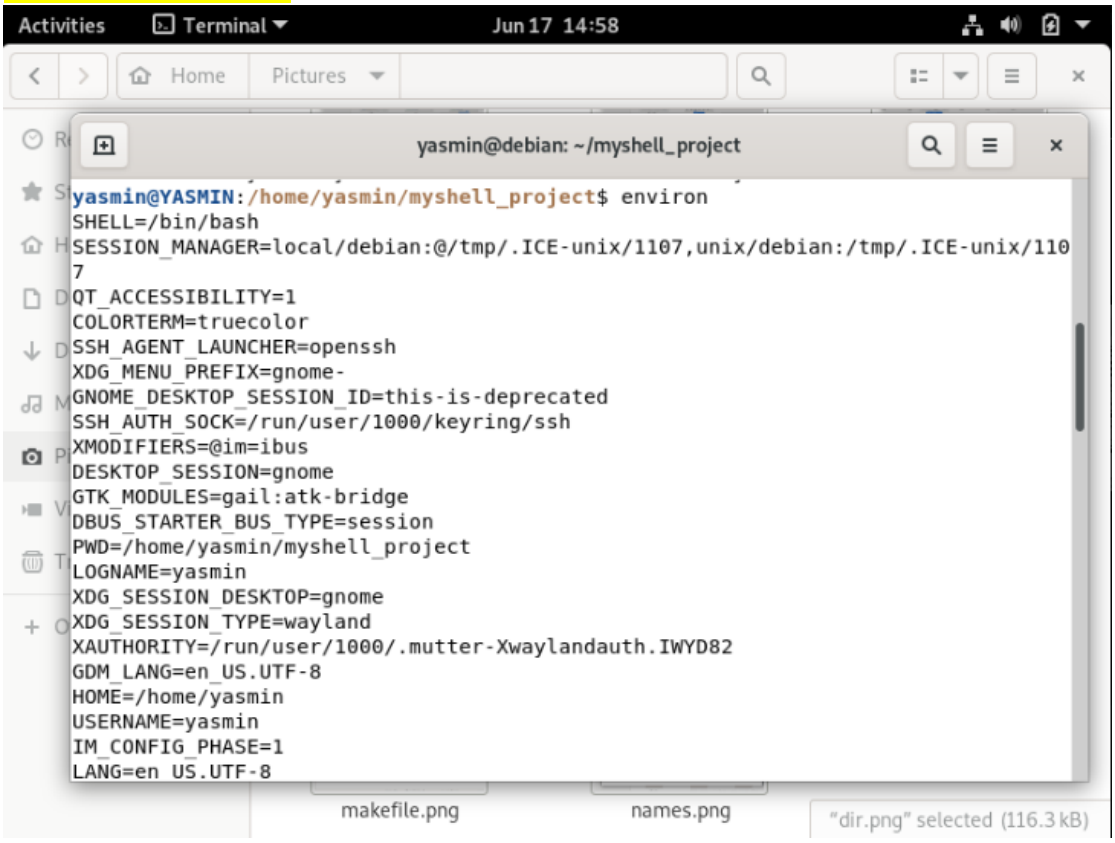
dir command :

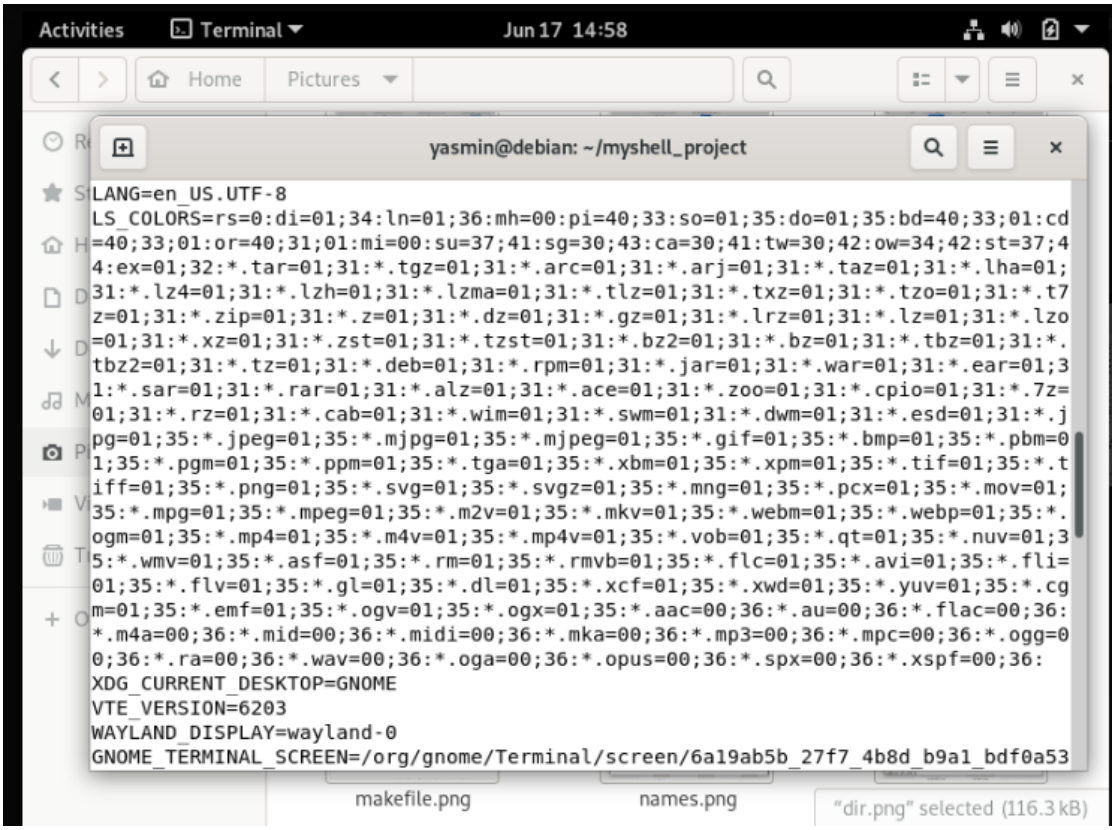


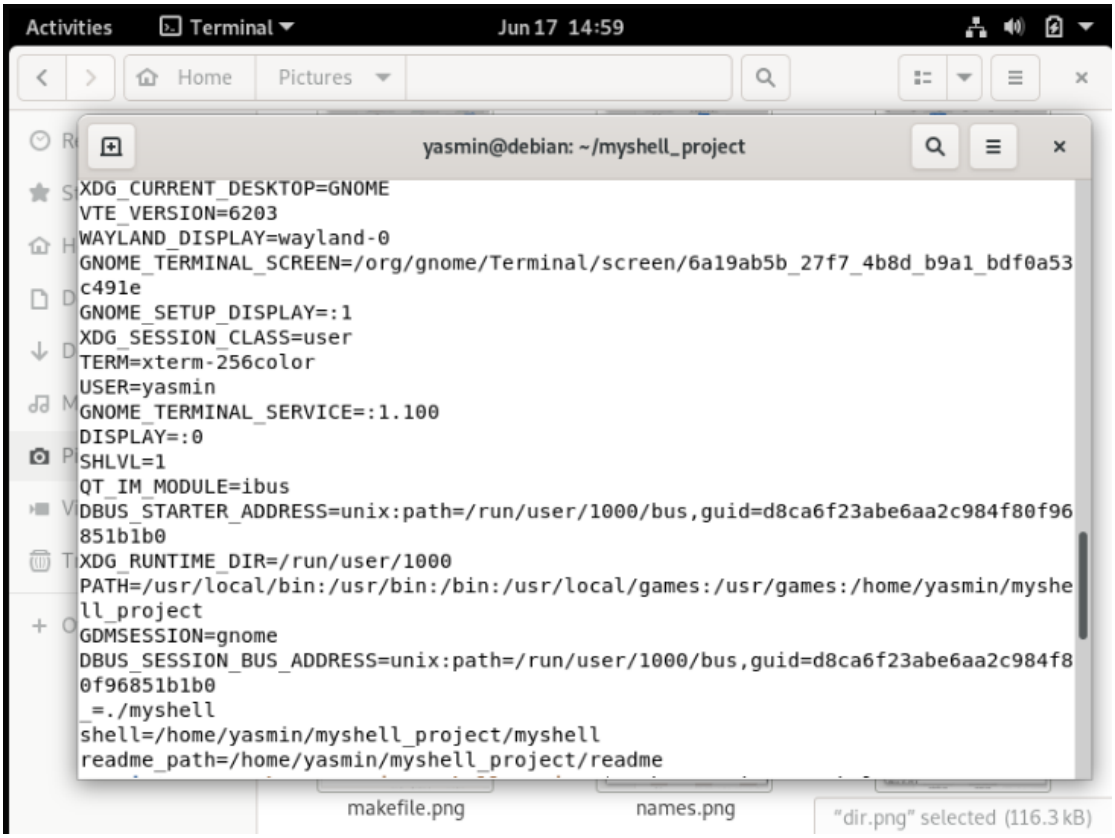
echo , ls ,ls -1 pause commands:



environ command:



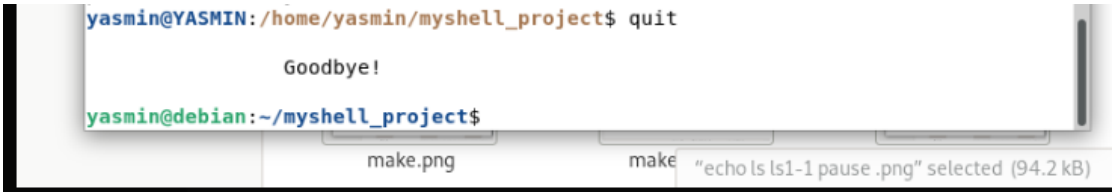




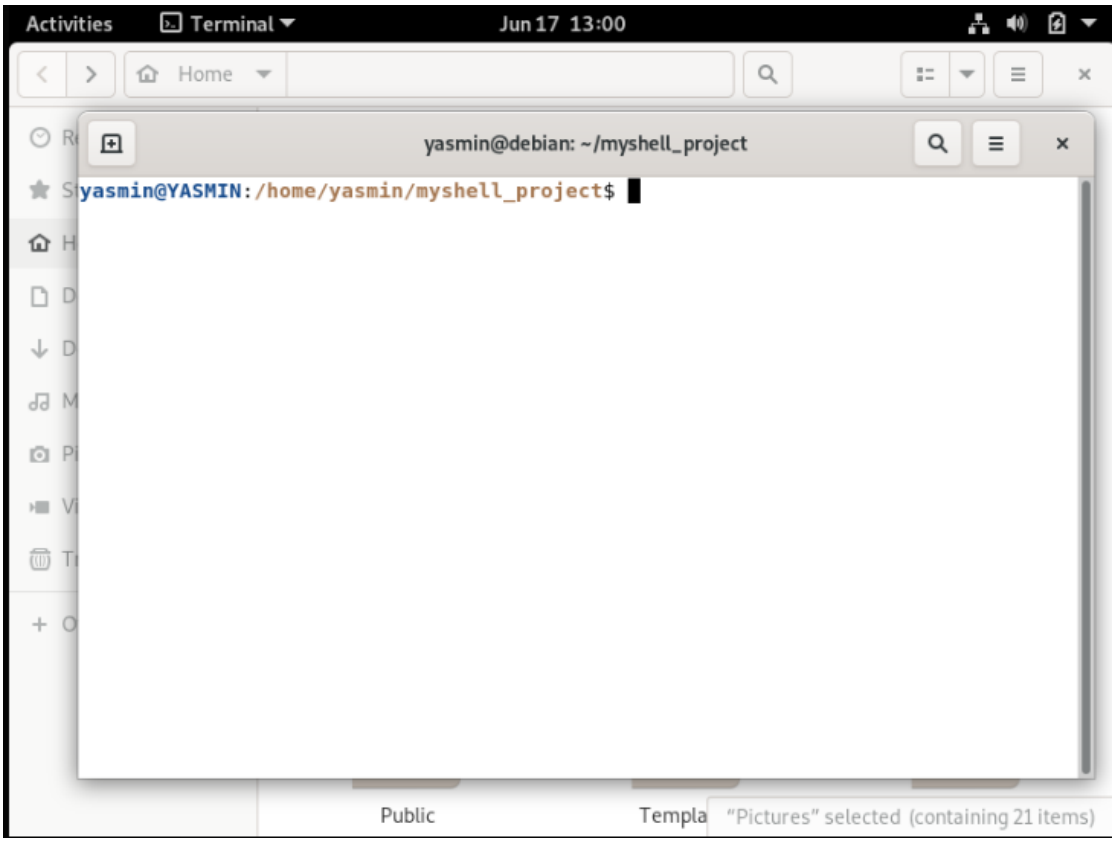
Help command:

```
yasmin@YASMIN:/home/yasmin/myshell_project$ help

myshell supports the following commands:

    cd    clear      dir    echo  environ

    help  myshell  pause  pwd   quit


An executable external file can also contain these commands.
To view the entire user manual, type "help more". To see information about a specific f
unction, type in:

"help help" ---see the function of 'help' command;

"help command" ---see the list of internal commands;

"help shell"---show more detail about this shell;

"help bat"---see  how the shell execute a batchfile;

"help i/o redirection"---learn about the i/o redirection;

"help background"---learn about the background mode;

"help path"---see the format of filepath or dir path.

yasmin@YASMIN:/home/yasmin/myshell_project$
```

quit command:



Command clear



Testing Redirection Operators

Output Redirection >

Command: echo text > output.txt

Result: Used cat to confirm correct output inside the file.

Output Append >>

Command: echo Free Gaza >> output.txt

Result: Appended content appeared in the file, as expected.

Input Redirection <

Command: File with echo Free Palestine Forever

Result: Shell read and executed the command successfully.

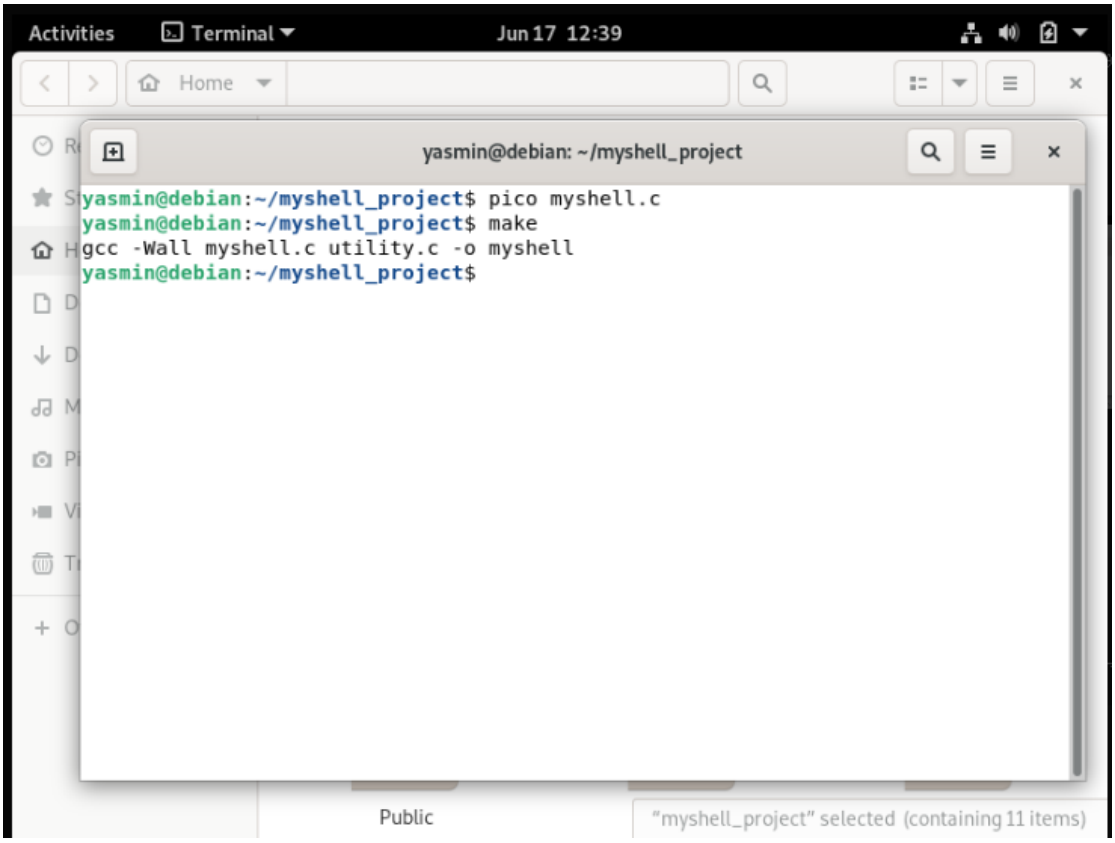
Testing Background Execution

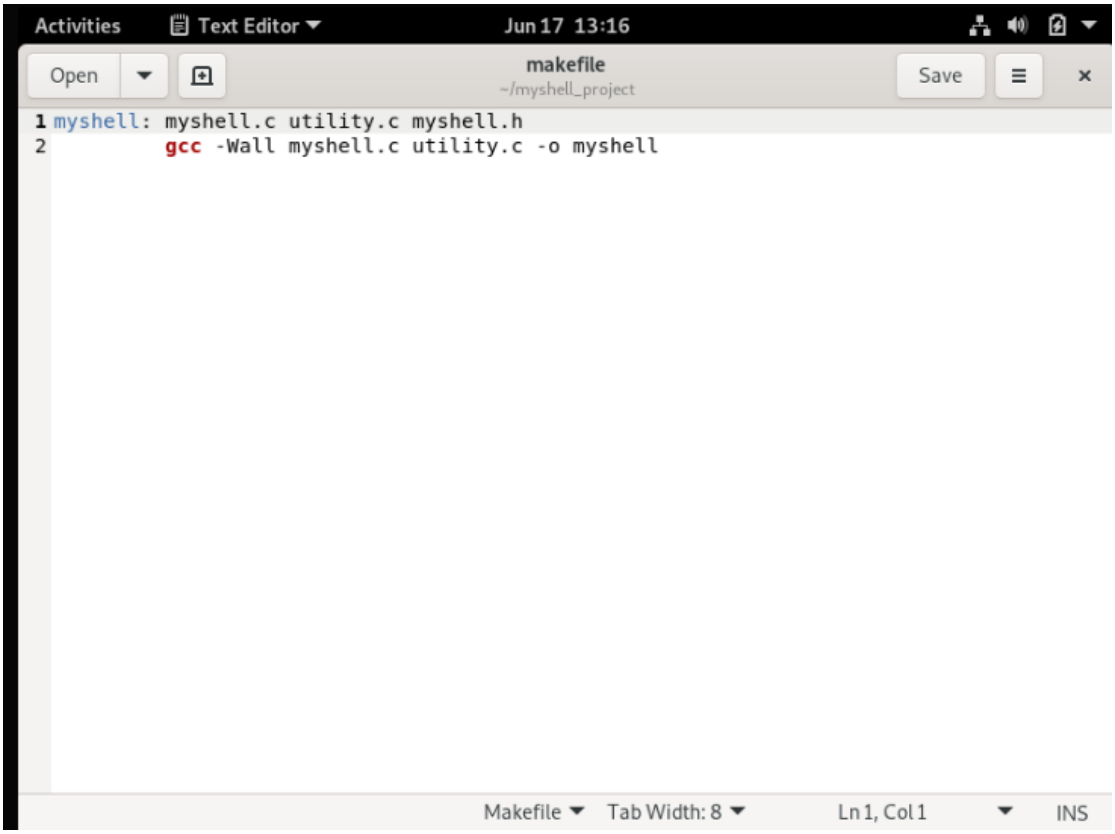
Background Process &

Command: sleep 5 &

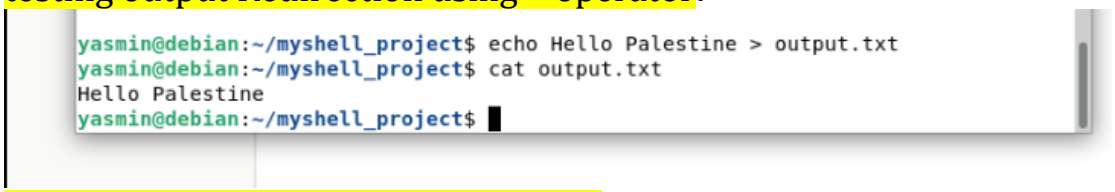
Result: Shell immediately returned a job number and PID, proving non-blocking execution.

makefile

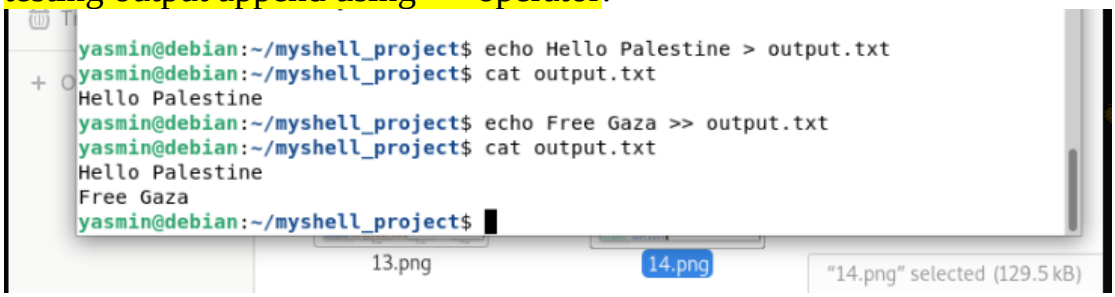




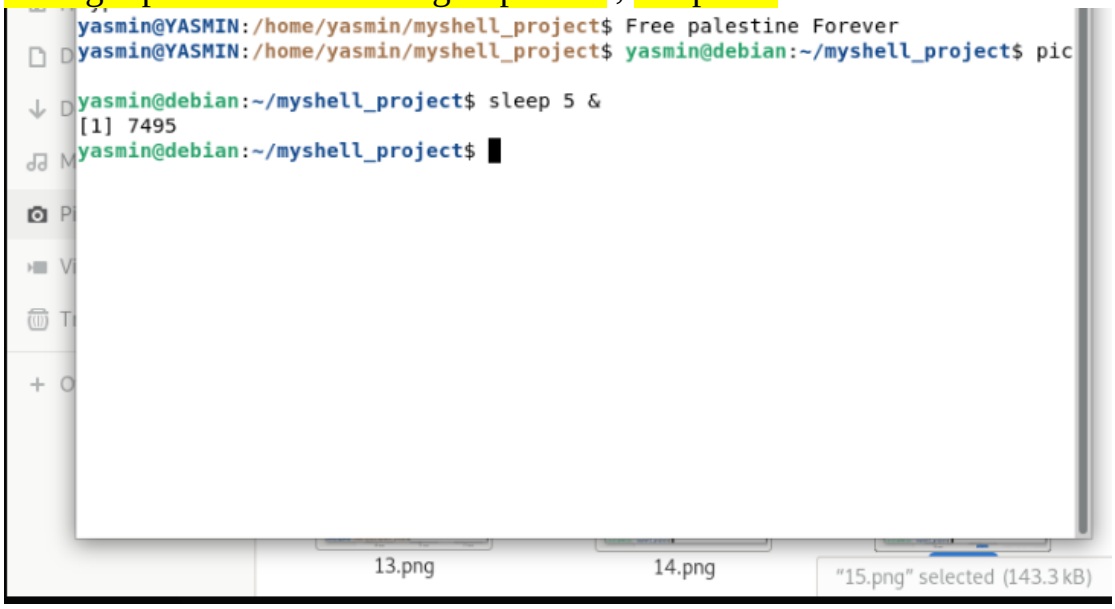
testing output Redirection using > operator:

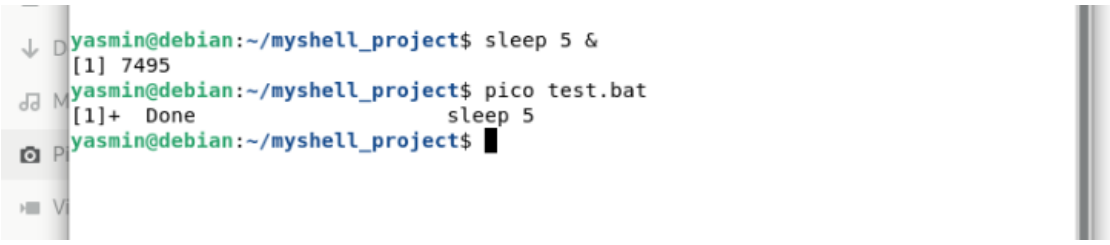


testing output append using >> operator:



testing input redirection using <operator , sleep 5 &





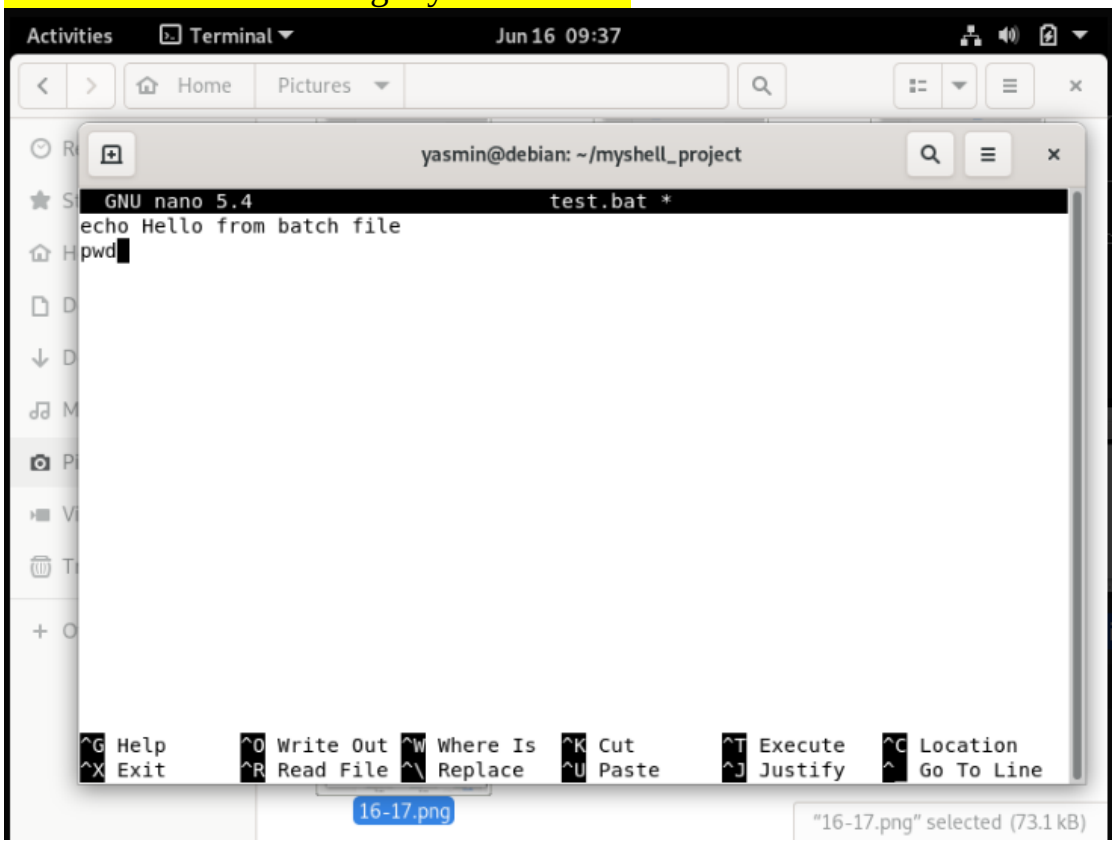
Testing Batch File Execution

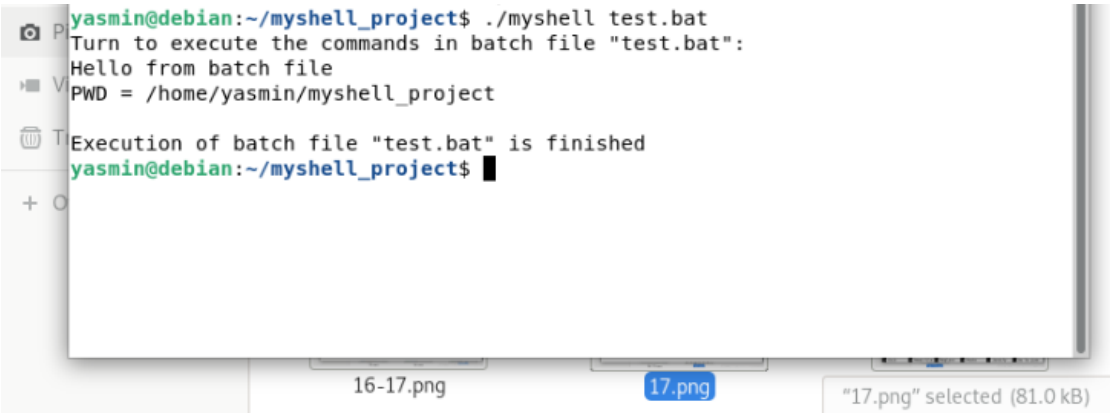
Creating batch file using pico test.bat.

Execution command: ./myshell test.bat

All commands in the file were executed sequentially, confirming batch processing support.

batch file execution using myshell file.bat





Conclusion

This project helped enhance our understanding of process control, file handling, and command parsing in Unix-based systems. It also provided practical experience in working with system calls and building modular C programs.