



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 :

```
1 #include "myshell.h"
2
3     /* Colors */
4 #define ANSI_COLOR_BRIGHT_RED "\033[1m\033[31m"
5 #define ANSI_COLOR_BIRGHT_BLUE "\033[1m\033[34m"
6 #define ANSI_COLOR_CYAN_BOLD "\033[1m\033[36m"
7 #define ANSI_COLOR_YELLOW_BOLD "\033[1m\033[33m"
8 #define ANSI_COLOR_RESET "\033[0m"
9
10
11 int main (int argc, char *argv[])
12 {
13
14     char buf[MAX_BUFFER], pwd[MAX_ARGS];    // line buffer
15     char shell_path[MAX_ARGS]="shell=";
16     char readme_path[MAX_ARGS]="readme_path=";
17     char newpath[MAX_ARGS*1000];
18     int len;
19
20     strcpy(newpath, getenv("PATH"));
21     strcat(newpath, ":");
22     if(strcmp(argv[0], "./myshell") && strcmp(argv[0], "myshell"))
23     {
24         len=strlen(argv[0]);
25         while(len&&argv[0][len]!='/')
26             len--;
27         argv[0][len]='\0';
28         strcpy(pwd, argv[0]);
29         get_fullpath(pwd, argv[0]);
30         printf("%s\n", pwd);
31     }
32     else
33         strcpy(pwd, getenv("PWD"));
34
35
36     }
37     else
38         strcpy(pwd, getenv("PWD"));
39
40     strcat(newpath, pwd);    // strcat(newpath, getenv("PWD"));
41     setenv("PATH", newpath, 1); // add the current working directory in the
42     "PATH" environment variable to search for the filename specified.
43     strcat(shell_path, pwd);    // strcat(shell_path, getenv("PWD"));
44     strcat(shell_path, "/myshell");
45     putenv(shell_path); //add the working directory of myshell in the
46     environment variables
47     strcat(readme_path, pwd);
48     strcat(readme_path, "/readme");
49     putenv(readme_path); // add the filepath of the file "readme" in the
50     environment variables, see function my_help( ) !
51
52     if(argc>1) // User input directly from the terminal ./myshell a.bat
53     >c.txt
54     {
55         strcpy(buf, "myshell ");
56         int i;
57         for(i=1; i<argc; i++)
58         {
59             strcat(buf, argv[i]);
60             strcat(buf, " ");
61         }
62         Execute(buf); // execute this command(bat) f~
63     }
64     else // if user input ./myshell
65     {
66         Command_clear( );
67         //print team members names
68     }
```

```

48         {
49             strcat(buf,argv[i]);
50             strcat(buf," ");
51         }
52     }
53     Execute(buf);// execute this command(bat)f~
54 }
55
56 else // if user input ./myshell
57 {
58
59     Command_clear( );
60     //print team members names
61     printf("\033[1;32m");
62     printf("Team Members:\n");
63     printf("-Yasmin Abdelaal\n");
64     printf("-Mona Alhasani\n");
65     printf("-Hala Abu Zerieq\n");
66     printf("\033[0m\n");
67     fprintf(stderr, ANSI_COLOR_CYAN_BOLD "Welcome to this simple shell!
68 \n" ANSI_COLOR_RESET);
69
70     printf("Type ");
71     printf(ANSI_COLOR_BRIGHT_BLUE "help" ANSI_COLOR_RESET);
72     printf(" to view manual\n");
73
74     printf("Type ");
75     printf(ANSI_COLOR_BRIGHT_RED "exit" ANSI_COLOR_RESET);
76     printf(" to terminate.\n");
77
78     Command_shell(stdin,NULL,NULL);
79 }
80 return 0 ;
81 } // end function "main"

```

code of myshell.h

```

1 #include <string.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/wait.h>
7 #include <errno.h>
8 #include <dirent.h>
9 #include <ctype.h>
10
11
12 #define MAX_BUFFER 1024 // max line buffer
13 #define MAX_ARGS 64 // max # args
14 #define SEPARATORS " \t\n" // token sparators
15 #define NUM 10 // the command number
16 #define MAX_OPEN 10 // open 10 stdin redirection files max and 10 stdout
17 // redirection files, myshell support several I/O redirectionf©
18 #define MAX_PATH 100 // the maxium length of file and forder
19
20
21 /*****
22 announce certain data types
23 *****/
24 typedef struct// redirection date structure
25 {
26     char *filename; // redirection file name
27     char opentype[3]; // the open method of redirection files "a" "r" "w"
28     char open[3]; // the open method of redirection files ">>" "<" ">"
29 } Redirect; //
30
31 extern int errno; // system error number
32 extern char **environ; // environment array
33

```

```

30
31 extern int errno;          // system error number
32 extern char **environ;     // environment array
33
34
35 /*****
36 announce the functions
37 *****/
38
39 int Execute(char *buffer); // the execution of the command
40 int Command_exec(char **args, const Redirect *Inputs, const Redirect *Outputs, int
    *states); //
41 int Error(int errortype, char **args, const Redirect * IOputs, const int *states, char
    * msg); // error message printout
42 int Command_strtok(char *buf, char **args, int *states, Redirect *InPuts, Redirect
    *OutPuts); // analyse the command
43 int Command_cd(char **args, const Redirect *Inputs, int *states); // execute command
    'cd'
44 void Command_clear(void); // execute command 'clear'
45 int Command_dir(char **args, const Redirect *Inputs, int *states); // execute command
    'dir'
46 int Command_echo(char **args, const Redirect *Inputs, int *states); // execute command
    'echo'
47 int list_envron(void); // execute the command "envron"
48 int show_pwd(void); // execute command 'pwd'
49 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.
50 void Command_delay(int n); // delay to ensure the order of processes
51 void get_fullpath(char *fullpath, const char *shortpath); // get the full path of a
    file or a directory
52 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 '#'

```

```

32 extern char **environ; // environment array
33
34
35 /*****
36 announce the functions
37 *****/
38
39 int Execute(char *buffer); // the execution of the command
40 int Command_exec(char **args, const Redirect *Inputs, const Redirect *Outputs, int
    *states); //
41 int Error(int errortype, char **args, const Redirect * IOputs, const int *states, char
    * msg); // error message printout
42 int Command_strtok(char *buf, char **args, int *states, Redirect *InPuts, Redirect
    *OutPuts); // analyse the command
43 int Command_cd(char **args, const Redirect *Inputs, int *states); // execute command
    'cd'
44 void Command_clear(void); // execute command 'clear'
45 int Command_dir(char **args, const Redirect *Inputs, int *states); // execute command
    'dir'
46 int Command_echo(char **args, const Redirect *Inputs, int *states); // execute command
    'echo'
47 int list_envron(void); // execute the command "envron"
48 int show_pwd(void); // execute command 'pwd'
49 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.
50 void Command_delay(int n); // delay to ensure the order of processes
51 void get_fullpath(char *fullpath, const char *shortpath); // get the full path of a
    file or a directory
52 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 '#'
53 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.

```
1 #include "myshell.h"
2
3 /* Colors */
4 #define ANSI_COLOR_BRIGHT_RED "\033[1m\033[31m"
5 #define ANSI_COLOR_BIRGHT_BLUE "\033[1m\033[34m"
6 #define ANSI_COLOR_CYAN_BOLD "\033[1m\033[36m"
7 #define ANSI_COLOR_YELLOW_BOLD "\033[1m\033[33m"
8 #define ANSI_COLOR_RESET "\033[0m"
9 // resets the text color to the default color for the terminal.
10
11 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
12 int output_num = 0; // used to store the number of output batch
    numbers in redirection., the number of output batch number in redirection
13 char batchfile[MAX_PATH]; // used to store the name of the current batch file.,
14 int bat_num = 0; // used to store the number of batch command-
    lines,
15 int isBatch = 0; // used to indicate whether the current process
    is a batch process 1 = yes , 0 = not
16 int letter; // store a single letter.
17 char *open; // store the name of a file that is to
    be opened
18
19 /* Function "Execute" : used as the function system( ), it interpret the input,
    and call function "Command_exec" to execute the all the commands. */
20 int Execute(char *buffer) // buffer contain a command to be executed
21 {
22     pid_t pid;
23     char *args[MAX_ARGS]; /// pointers to arg strings
24     int error;
25     int states[5]; // states[0] is back exec; states[1] is inputfile num;
    states[2] is outoutfile num :
```

```

25     int states[5]; // states[0] is back exec; states[1] is inputfile num;
states[2] is outputfile num ;
26     // states[3] is priority of inputfile(not args); states[4] is argc ;
27     Redirect Inputs[MAX_OPEN]; // input redirection (10) max files
can be opened at once
28     Redirect Outputs[MAX_OPEN]; // output redirection (10)
29
30     error = Command_strtok(buffer, args, states, Inputs, Outputs);
31     /*
32     تقسم النص حسب التوكنايز
33     tokenizing the command in buffer into individual arguments,
34     and storing the resulting array of strings in args.
35     It also populates the states array and the Inputs and Outputs arrays
36     with information about the command.
37     :tokenize input last entry will be NULL
38     */
39     if (error || args[0] == NULL)
40         return -1; // If the input format error or if there's anything
there
41
42     if (!strcmp(args[0], "quit") || !strcmp(args[0], "exit")) // if entered
"quit" command
43     {
44         //It checks if there is an argument after the quit command.
45         if
(args[1]) // no
argument is needed after "quit"
46             Error(-2, args + 1, NULL, NULL, args[0]);
47
48         if (output_num > 1) // e.g. myshell test.bat >m.txt >n.txt
49         {
50             fprintf(stderr, "Exit\n");
51             return 1;

```

```

51             return 1;
52         }
53
54         if (isBatch)
55             fprintf(stderr, "Batch file \"%s\" is finished!\n",
batchfile);
56         else
57             fprintf(stderr, "\n\t\tGoodbye! \n\n");
58         exit(0); // break out of 'while (!feof(stdin))' loop in "main"
59     }
60
61     else if (states[0]) // states[0]==1 running in background "flag indicate
whether executed in background or not"
62     {
63         switch (pid = fork())
64         {
65             case -1:
66                 Error(-9, NULL, NULL, states, "fork");//print error
message and exit prog
67             case 0: // child //sleep(1);
68                 Command_delay(12);
69                 fprintf(stderr, "\n");
70                 Command_exec(args, Inputs, Outputs, states);
71                 exit(1);
72             default:
73                 if (isBatch == 0)
74                     fprintf(stderr, "pid=%d\n", pid);
75             } // end switch
76         }
77
78         else // states[0]==0 running in front
79             Command_exec(args, Inputs, Outputs, states);

```

```

80 |
81 |         return 0;
82 |     }
83 |
84 | /* Function "Command_exec" : execute the command */
85 | int Command_exec(char **args, const Redirect *Inputs, const Redirect *Outputs, int
    *states)
86 | {
87 |     char filepath[MAX_PATH], parent[MAX_ARGS];
88 |     FILE *outputfile = NULL, *inputfile;
89 |     pid_t newpid;
90 |     int flag;
91 |
92 |     if (!strcmp(args[0], "myshell") || !strcmp(args[0], "shell")) // strcmp()
    cmp function "myshell" command, checks if the first argument s either "myshell" or
    "shell"
93 |     {
94 |         flag = 0;
95 |         if (isBatch) // e.g. execute myshell b.txt in test.bat
96 |         {
97 |             switch (newpid = fork())
98 |             {
99 |                 case -1:
100 |                     Error(-9, NULL, NULL, states, "fork");
101 |                 case 0:
102 |                     // is both a background process and a batch process.
103 |                     if (states[0] && (args[1] || states[1]))
104 |                     {
105 |                         back_bat++;
106 |                         flag = 1;
107 |                     }
108 |                     output_num = states[2];
109 |                     Command_bat(args, Inputs, Outputs, states);

```

```

109 |                     Command_bat(args, Inputs, Outputs, states);
110 |                     if (flag)
111 |                         back_bat--;
112 |                     output_num = 0;
113 |                     exit(0);
114 |                 default:
115 |                     waitpid(newpid, NULL, WUNTRACED);
116 |             }
117 |         }
118 |         else
119 |         {
120 |             if (states[0] && (args[1] || states[1]))
121 |             {
122 |                 back_bat++;
123 |                 flag = 1;
124 |             }
125 |             output_num = states[2];
126 |             Command_bat(args, Inputs, Outputs, states);
127 |             if (flag)
128 |                 back_bat--;
129 |             output_num = 0;
130 |         }
131 |         if (states[0])
132 |             exit(1);
133 |         else
134 |             return 0;
135 |     }
136 |
137 |     if (states[2]) // set output Redirection : use freopen()
138 |     {
139 |         get_fullpath(filepath, Outputs->filename);
140 |         outputfile = freopen(filepath, Outputs->opentype, stdout);
141 |         if (outputfile == NULL)

```



```

141         if (outputfile == NULL)
142         {
143             Error(-6, NULL, NULL, NULL, Outputs->filename);
144             if (states[0])
145                 exit(1);
146             else
147                 return -4;
148         }
149     }
150
151     // check for internal/external command
152     if (!strcmp(args[0], "cd")) // "cd " command
153         Command_cd(args, Inputs, states);
154
155     else if (strcmp(args[0], "clr") == 0 || strcmp(args[0], "clear") == 0) //
156     "clear" command
157     {
158         system("clear");
159         if (output_num == 0)
160             Command_clear(); // In
161         Command_clear(), execute clear
162         if (args[1] || states[1] || states[2]) // no argument is needed
163             Error(4, NULL, NULL, NULL, args[0]);
164     }
165
166     else if (!strcmp(args[0], "dir")) // "dir" command
167         Command_dir(args, Inputs, states);
168
169     else if (!strcmp(args[0], "echo")) // "echo" command
170         Command_echo(args, Inputs, states); //

```

makefile

The screenshot shows a text editor window titled 'makefile' with the path '~/myshell_project'. The editor contains the following content:

```

1 myshell: myshell.c utility.c myshell.h
2 gcc -Wall myshell.c utility.c -o myshell

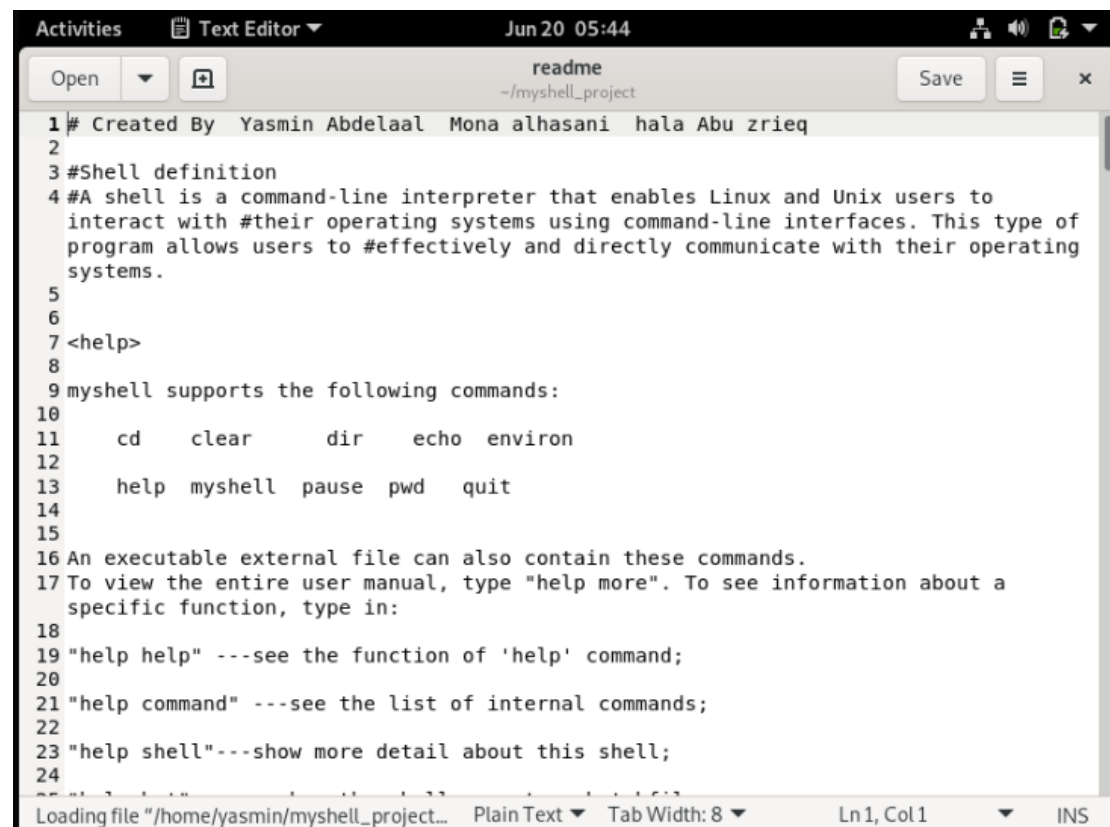
```

The status bar at the bottom indicates 'Makefile', 'Tab Width: 8', 'Ln 1, Col 1', and '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.



The screenshot shows a text editor window titled 'readme' with the path '~/.myshell_project'. The window contains the following text:

```
1 # Created By Yasmin Abdelaal Mona alhasani hala Abu zrieq
2
3 #Shell definition
4 #A shell is a command-line interpreter that enables Linux and Unix users to
  interact with #their operating systems using command-line interfaces. This type of
  program allows users to #effectively and directly communicate with their operating
  systems.
5
6
7 <help>
8
9 myshell supports the following commands:
10
11     cd    clear    dir    echo  environ
12
13     help  myshell  pause  pwd   quit
14
15
16 An executable external file can also contain these commands.
17 To view the entire user manual, type "help more". To see information about a
  specific function, type in:
18
19 "help help" ---see the function of 'help' command;
20
21 "help command" ---see the list of internal commands;
22
23 "help shell"---show more detail about this shell;
24
```

The status bar at the bottom indicates the file path as '/home/yasmin/myshell_project...', the text format as 'Plain Text', the tab width as 8, and the cursor position as 'Ln 1, Col 1'.

Activities Text Editor Jun 20 05:45

Open readme Save

~/myshell_project

```
24
25 "help bat"---see how the shell execute a batchfile;
26
27 "help i/o redirection"---learn about the i/o redirection;
28
29 "help background"---learn about the background mode;
30
31 "help path"---see the format of filepath or dir path.
32
33 #
34
35
36
37
38
39 <help shell>
40
41 To view more details about this shell;
42
43 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.
44 The shell process involves the following steps:
45
46 * interprets the input,
47
48 * takes appropriate action, and
```

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

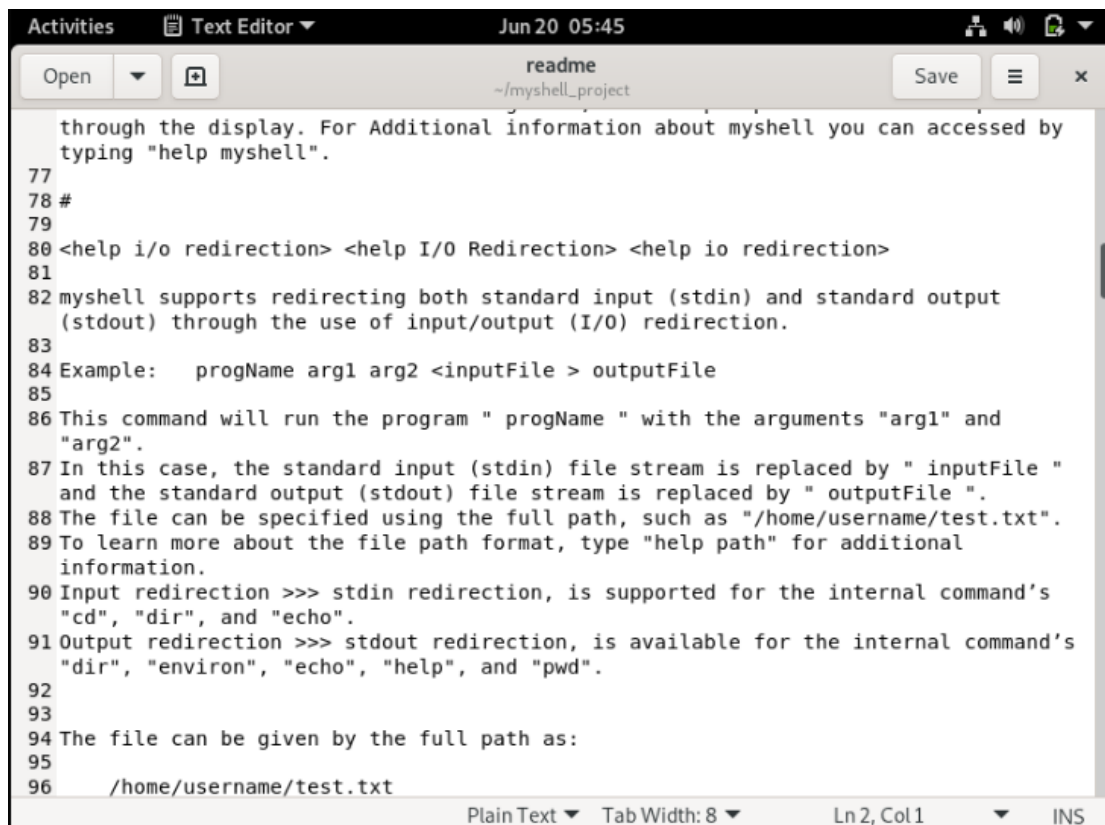
Activities Text Editor Jun 20 05:45

Open readme Save

~/myshell_project

```
48 * takes appropriate action, and
49
50 * finally prompts for more input.
51 #
52
53 <help command> <help commands> <help internal command> <help internal commands>
54 To view a list of internal commands:
55 To access detailed usage information for each command, write "help [command]"
56 Example: help cd
57
58 myshell supports the following commands:
59
60 cd clr dir echo environ
61
62 help myshell pause pwd quit
63
64 #
65
66
67
68 <help bat> <help batchfile>
69
70
71 myshell is capable of obtaining its command line input from a batch file.
72 If myshell is invoked with a command line argument:
73
74 Example: myshell test.bat
75
76 the batch file "test.bat" is assumed to contain a series of commands for the shell
```

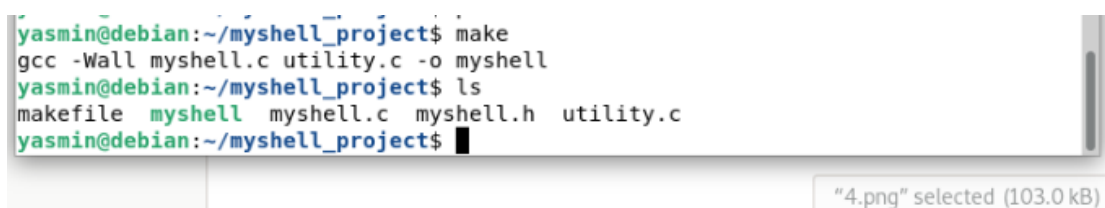
Plain Text Tab Width: 8 Ln 2, Col 1 INS

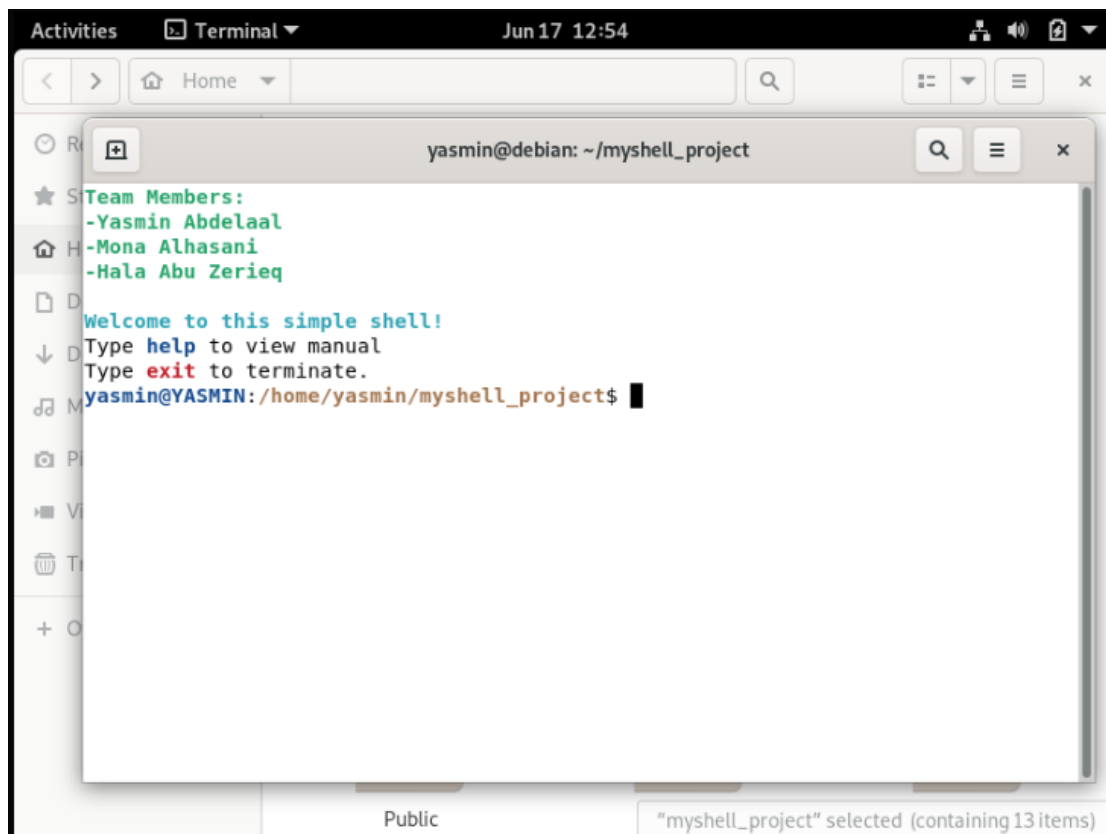


PROJECT OUTPUT:

Running the Custom Shell Successfully

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





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

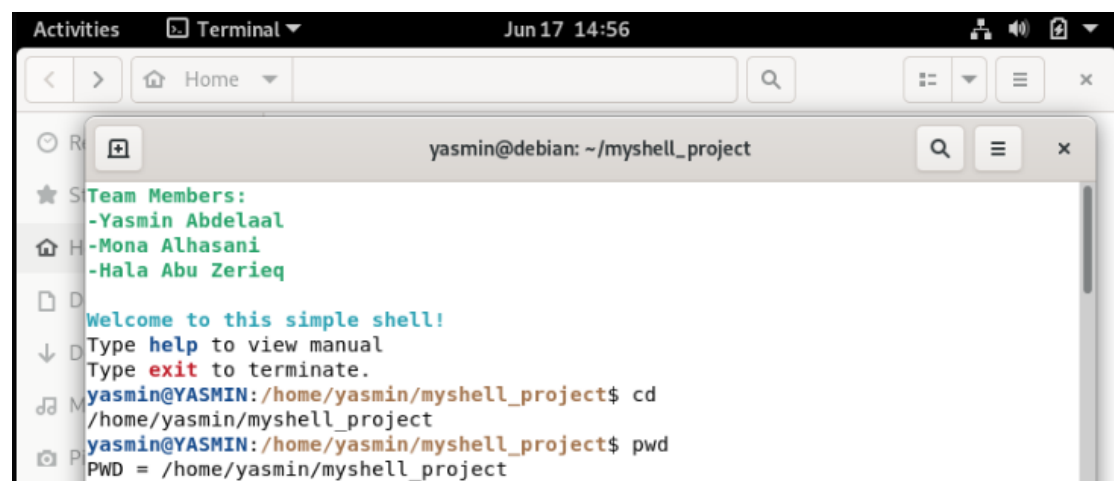
Result: Successfully read help content from file and displayed it, confirming string matching and file reading logic.

quit Command

Purpose: Exit shell

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

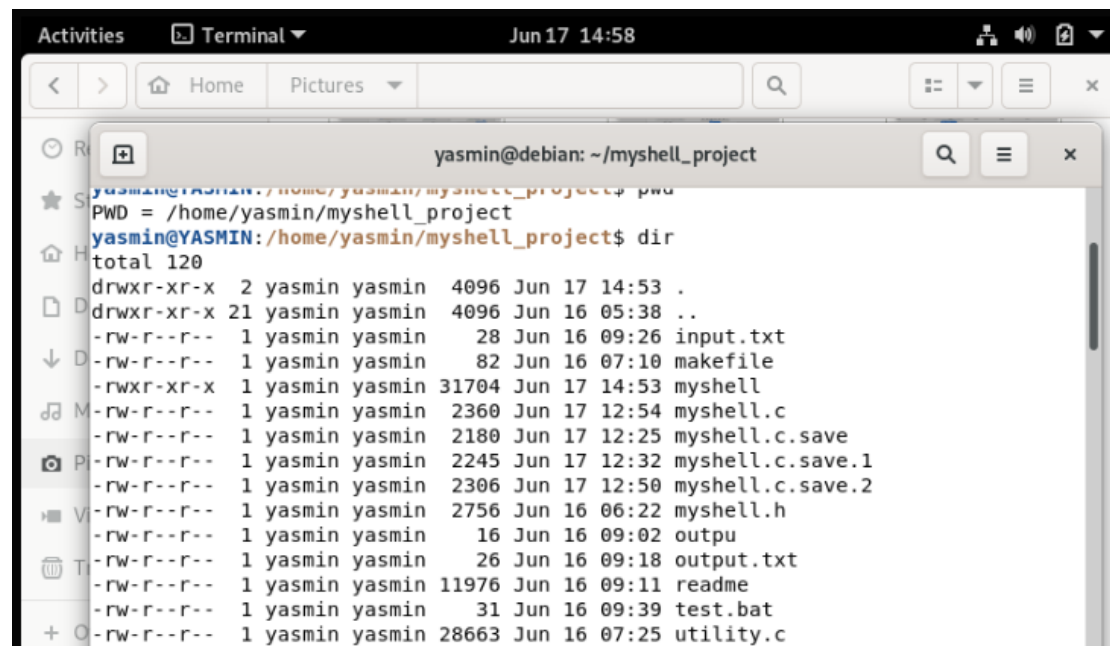
Cd , pwd commands:



The screenshot shows a terminal window titled 'Terminal' with the date and time 'Jun 17 14:56'. The terminal prompt is 'yasmin@debian: ~/myshell_project'. The output of the 'cd' command is '/home/yasmin/myshell_project'. The output of the 'pwd' command is '/home/yasmin/myshell_project'. The terminal also displays a welcome message and instructions for using the shell.

```
Activities Terminal Jun 17 14:56
yasmin@debian: ~/myshell_project
Team Members:
-Yasmin Abdelaal
-Mona Alhasani
-Hala Abu Zerieq
Welcome to this simple shell!
Type help to view manual
Type exit to terminate.
yasmin@YASMIN:/home/yasmin/myshell_project$ cd
/home/yasmin/myshell_project
yasmin@YASMIN:/home/yasmin/myshell_project$ pwd
PWD = /home/yasmin/myshell_project
```

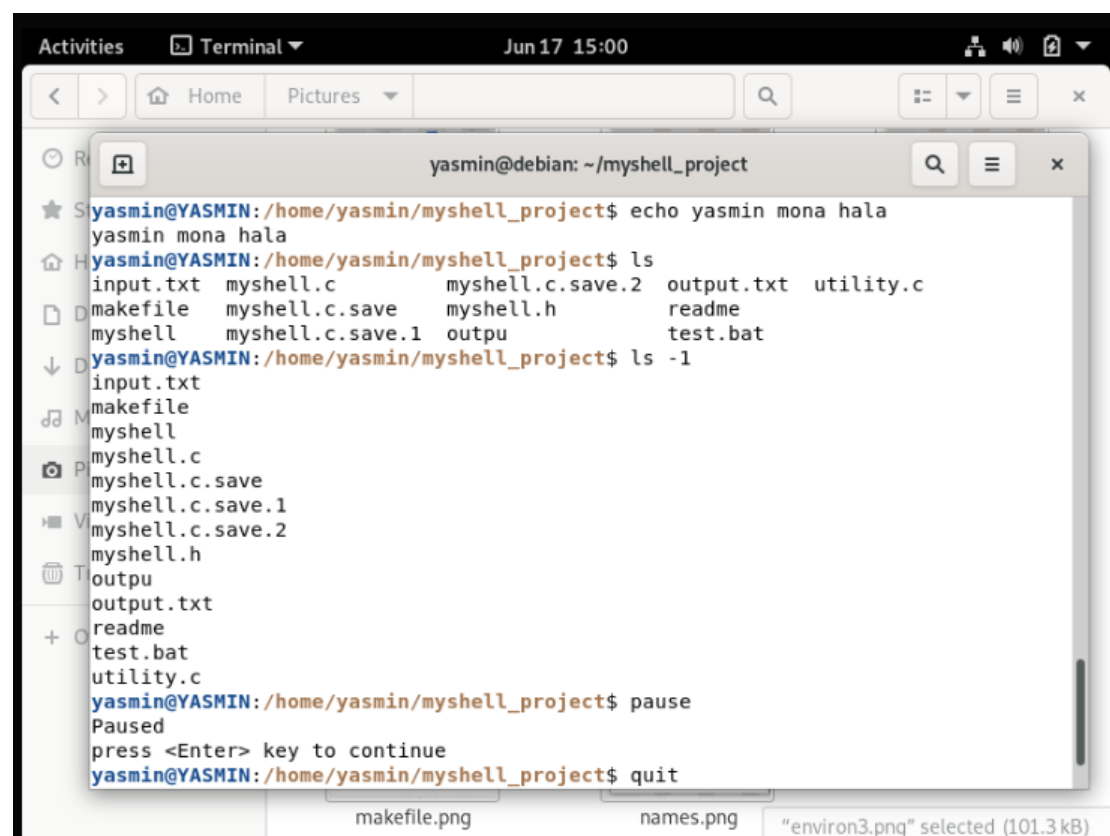
dir command :



A terminal window titled 'yasmin@debian: ~/myshell_project' showing the output of the 'dir' command. The output lists files and directories with their permissions, owner, group, size, and date. The files are: ., .., input.txt, makefile, myshell, myshell.c, myshell.c.save, myshell.c.save.1, myshell.c.save.2, myshell.h, output, output.txt, readme, test.bat, and utility.c.

```
yasmin@debian: ~/myshell_project
yasmin@YASMIN:/home/yasmin/myshell_project$ pwd
PWD = /home/yasmin/myshell_project
yasmin@YASMIN:/home/yasmin/myshell_project$ dir
total 120
drwxr-xr-x  2 yasmin yasmin  4096 Jun 17 14:53 .
drwxr-xr-x 21 yasmin yasmin  4096 Jun 16 05:38 ..
-rw-r--r--  1 yasmin yasmin   28 Jun 16 09:26 input.txt
-rw-r--r--  1 yasmin yasmin   82 Jun 16 07:10 makefile
-rwxr-xr-x  1 yasmin yasmin 31704 Jun 17 14:53 myshell
-rw-r--r--  1 yasmin yasmin  2360 Jun 17 12:54 myshell.c
-rw-r--r--  1 yasmin yasmin  2180 Jun 17 12:25 myshell.c.save
-rw-r--r--  1 yasmin yasmin  2245 Jun 17 12:32 myshell.c.save.1
-rw-r--r--  1 yasmin yasmin  2306 Jun 17 12:50 myshell.c.save.2
-rw-r--r--  1 yasmin yasmin  2756 Jun 16 06:22 myshell.h
-rw-r--r--  1 yasmin yasmin   16 Jun 16 09:02 output
-rw-r--r--  1 yasmin yasmin   26 Jun 16 09:18 output.txt
-rw-r--r--  1 yasmin yasmin 11976 Jun 16 09:11 readme
-rw-r--r--  1 yasmin yasmin   31 Jun 16 09:39 test.bat
-rw-r--r--  1 yasmin yasmin 28663 Jun 16 07:25 utility.c
```

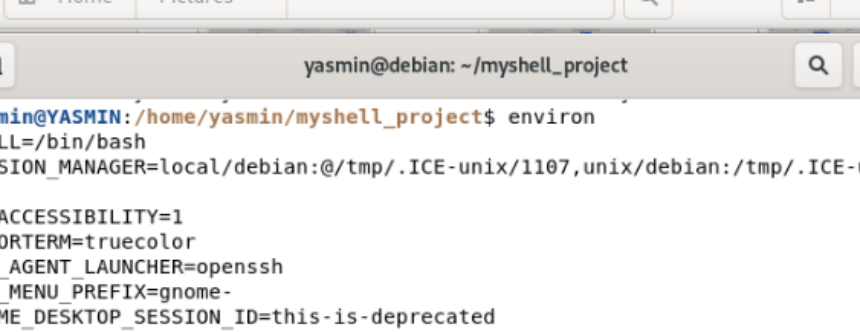
echo , ls ,ls -l pause commands:



A terminal window titled 'yasmin@debian: ~/myshell_project' showing the output of several commands. The commands and their outputs are: 'echo yasmin mona hala' outputs 'yasmin mona hala'; 'ls' outputs a single line of file names; 'ls -l' outputs a detailed listing of files; 'pause' outputs 'Paused' and 'press <Enter> key to continue'; 'quit' outputs nothing.

```
yasmin@debian: ~/myshell_project
yasmin@YASMIN:/home/yasmin/myshell_project$ echo yasmin mona hala
yasmin mona hala
yasmin@YASMIN:/home/yasmin/myshell_project$ ls
input.txt  myshell.c      myshell.c.save.2  output.txt  utility.c
makefile  myshell.c.save  myshell.h         readme
myshell   myshell.c.save.1  output           test.bat
yasmin@YASMIN:/home/yasmin/myshell_project$ ls -l
input.txt
makefile
myshell
myshell.c
myshell.c.save
myshell.c.save.1
myshell.c.save.2
myshell.h
output
output.txt
readme
test.bat
utility.c
yasmin@YASMIN:/home/yasmin/myshell_project$ pause
Paused
press <Enter> key to continue
yasmin@YASMIN:/home/yasmin/myshell_project$ quit
```

environ command:



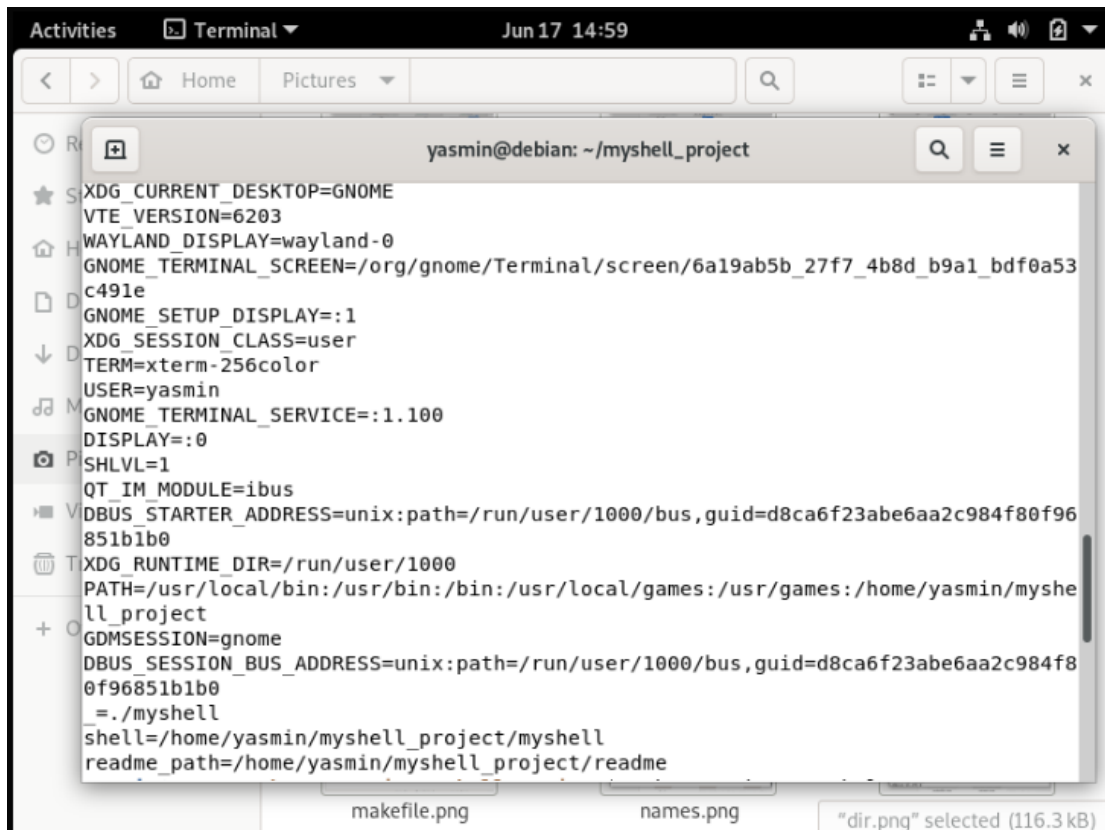
The screenshot shows a terminal window titled "Terminal" with the date and time "Jun 17 14:58". The terminal prompt is "yasmin@debian: ~/myshell_project". The user has entered the command "env", and the output lists various environment variables:

```
yasmin@YASMIN:/home/yasmin/myshell_project$ env  
SHELL=/bin/bash  
SESSION_MANAGER=local/debian:@/tmp/.ICE-unix/1107,unix/debian:/tmp/.ICE-unix/1107  
QT_ACCESSIBILITY=1  
COLORTERM=truecolor  
SSH_AGENT_LAUNCHER=openssh  
XDG_MENU_PREFIX=gnome-  
GNOME_DESKTOP_SESSION_ID=this-is-deprecated  
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh  
XMODIFIERS=@im=ibus  
DESKTOP_SESSION=gnome  
GTK_MODULES=gail:atk-bridge  
DBUS_STARTER_BUS_TYPE=session  
PWD=/home/yasmin/myshell_project  
LOGNAME=yasmin  
XDG_SESSION_DESKTOP=gnome  
XDG_SESSION_TYPE=wayland  
XAUTHORITY=/run/user/1000/.mutter-Xwaylandauth.IWYD82  
GDM_LANG=en_US.UTF-8  
HOME=/home/yasmin  
USERNAME=yasmin  
IM_CONFIG_PHASE=1  
LANG=en_US.UTF-8
```

The screenshot shows a terminal window titled "Terminal" with the date and time "Jun 17 14:58". The terminal prompt is "yasmin@debian: ~/myshell_project". The output of the command is a list of environment variables:

```
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd
=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;4
4:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;
31:*.lzh=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7
z=01;31:*.zip=01;31:*.z=01;31:*.d=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo
=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.
tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;3
1:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=
01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.j
pg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=0
1;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.t
iff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;
35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.webp=01;35:*.
ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;3
5:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=
01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cg
m=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:
*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=0
0;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
XDG_CURRENT_DESKTOP=GNOME
VTE_VERSION=6203
WAYLAND_DISPLAY=wayland-0
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/6a19ab5b_27f7_4b8d_b9a1_bdf0a53
```

At the bottom of the terminal window, there are three buttons: "makefile.png", "names.png", and "dir.png" selected (116.3 kB).



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 function, 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:

```
yasmin@YASMIN:/home/yasmin/myshell_project$ quit
Goodbye!
yasmin@debian:~/myshell_project$
```

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.

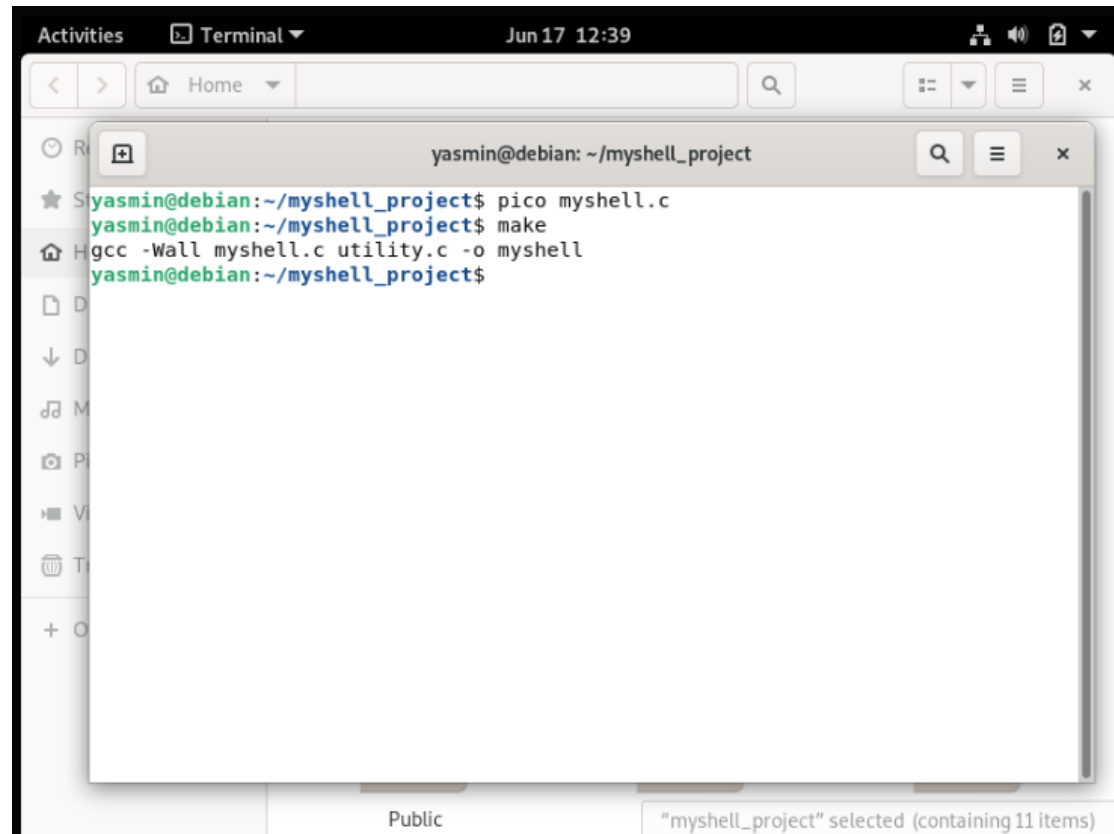
Testing Background Execution

Background Process &

Command: `sleep 5 &`

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

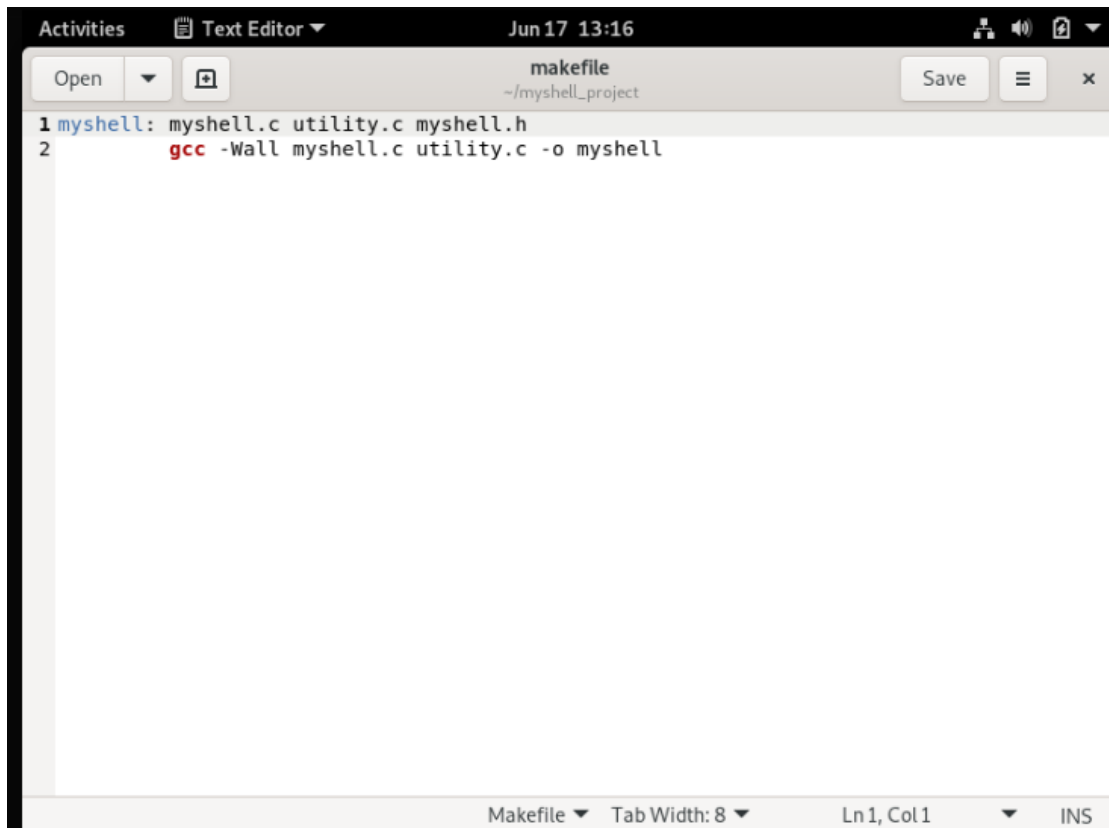
makefile



The screenshot shows a terminal window titled "yasmin@debian: ~/myshell_project". The terminal output is as follows:

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

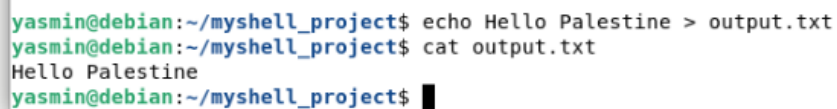
The terminal window is part of a desktop environment. The top bar shows "Activities", "Terminal", and the date "Jun 17 12:39". The bottom bar shows "Public" and a notification "myshell_project" selected (containing 11 items).



A screenshot of a text editor window titled "makefile" with the path "~/myshell_project". The window contains two lines of text: "1 myshell: myshell.c utility.c myshell.h" and "2 gcc -Wall myshell.c utility.c -o myshell". The status bar at the bottom indicates "Makefile", "Tab Width: 8", "Ln 1, Col 1", and "INS".

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


testing output Redirection using > operator:



A screenshot of a terminal window showing the following commands and output: "yasmin@debian:~/myshell_project\$ echo Hello Palestine > output.txt", "yasmin@debian:~/myshell_project\$ cat output.txt", and "Hello Palestine".

```
yasmin@debian:~/myshell_project$ echo Hello Palestine > output.txt
yasmin@debian:~/myshell_project$ cat output.txt
Hello Palestine
yasmin@debian:~/myshell_project$
```

testing output append using >> operator:



A screenshot of a terminal window showing the following commands and output: "yasmin@debian:~/myshell_project\$ echo Hello Palestine > output.txt", "yasmin@debian:~/myshell_project\$ cat output.txt", "Hello Palestine", "yasmin@debian:~/myshell_project\$ echo Free Gaza >> output.txt", "yasmin@debian:~/myshell_project\$ cat output.txt", "Hello Palestine", and "Free Gaza".

```
yasmin@debian:~/myshell_project$ echo Hello Palestine > output.txt
yasmin@debian:~/myshell_project$ cat output.txt
Hello Palestine
yasmin@debian:~/myshell_project$ echo Free Gaza >> output.txt
yasmin@debian:~/myshell_project$ cat output.txt
Hello Palestine
Free Gaza
yasmin@debian:~/myshell_project$
```

testing input redirection using <operator , sleep 5 &

```
yasmin@YASMIN:/home/yasmin/myshell_project$ Free palestine Forever
yasmin@YASMIN:/home/yasmin/myshell_project$ yasmin@debian:~/myshell_project$ pic
yasmin@debian:~/myshell_project$ sleep 5 &
[1] 7495
yasmin@debian:~/myshell_project$
```

```
yasmin@debian:~/myshell_project$ sleep 5 &
[1] 7495
yasmin@debian:~/myshell_project$ pico test.bat
[1]+  Done                  sleep 5
yasmin@debian:~/myshell_project$
```

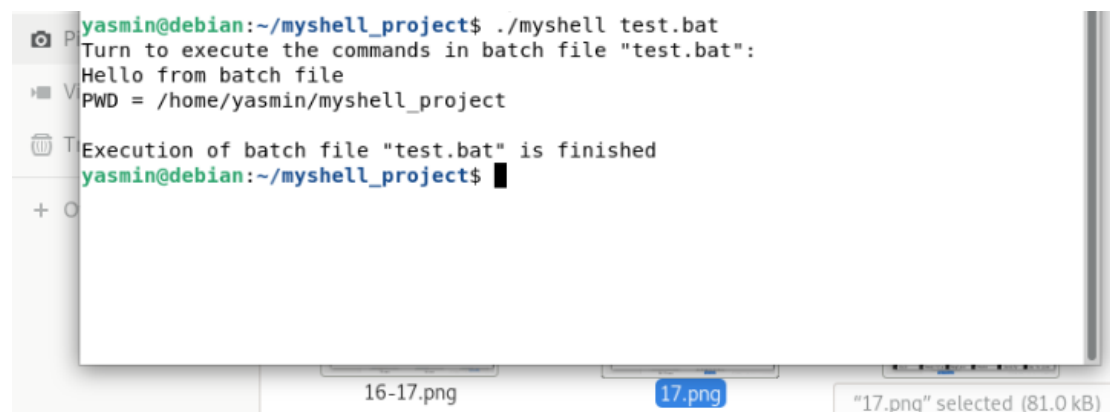
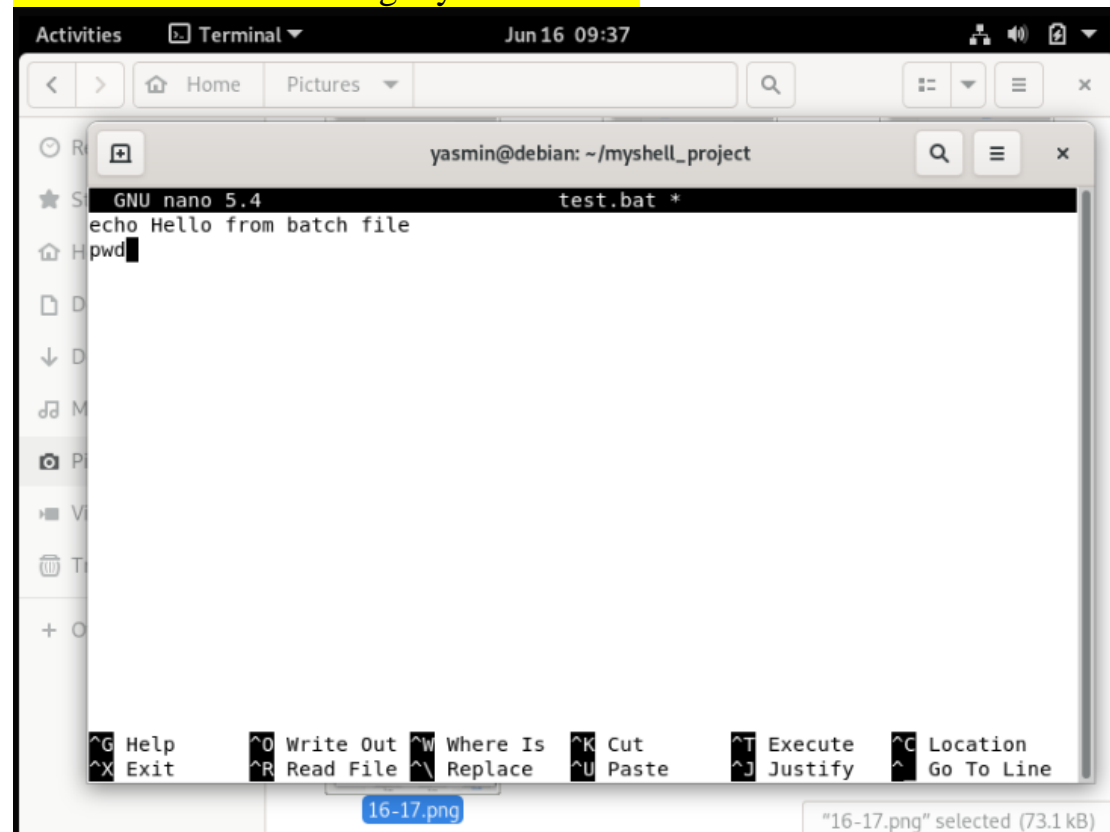
Testing Batch File Execution

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.

