
TERMINUS

May 18, 2020

University of the Basque Country (UPV/EHU)

Introduction to Operating Systems

Aitor Malo, Unai Fernandez, Haizea Garaño and Unai Salas

Contents

1	Introduction	3
2	Description	4
2.1	The shell	4
2.2	The server	4
2.2.1	Server routines	4
2.3	Commands	5
2.4	The game	5
3	Specifcation documents	6
3.1	cd	6
3.2	cp	7
3.3	grep	7
3.4	less	8
3.5	man	9
3.6	pwd	10
3.7	touch	10
3.8	mv	11
3.9	ls	12
3.10	sudo	12
4	Verification documents	14
4.1	Program or utility to verify: rm	14
4.2	Program or utility to verify: ls	14
4.3	Program or utility to verify: less	15
4.4	Program or utility to verify: pwd	16
4.5	Program or utility to verify: touch	16
4.6	Program or utility to verify: cp	17
4.7	Program or utility to verify: mv	18
4.8	Program or utility to verify: grep	19

4.9	Program or utility to verify: man	19
4.10	Program or utility to verify: sudo	20
5	Theoretical concepts used in the project	22
5.1	System calls	22
5.2	Commands	22
5.3	Functions & Libraries	22
5.4	Pipes	23
5.5	Permissions	23
5.6	Processes	24
6	Distribution	25
6.1	Unai Fernandez	25
6.2	Haizea Garaño	25
6.3	Aitor Malo	25
6.4	Unai Salas	25
7	Installation Guide	26
7.1	After the installation	26
8	Manual	27
9	Conclusions	28

1 Introduction

Terminus is a text-based adventure game, aimed at teaching the player to use the command line with basic commands. In this project the game has been replicated with code that we have created to put into practice the theory learned during the course.

In this project we have used these main commands: `cd`, `cp`, `grep`, `less`, `ls`, `man`, `pwd` and `touch`. Each command has its own documentation in the `man`. The client-server structure has also been used, applying the knowledge acquired theoretically about it

2 Description

This project consists of three main parts, there is a shell, program where the user will type all the commands and play the game. Then there is a server, this part will manage all the commands and perform certain actions depending on those. Finally we have the programs for each of the commands. All the command except *cd* should be executed individually.

2.1 The shell

The shell is the main program, one of the most important parts, it manages most of the actions of the game we created. Inside this main program we can find the *execute* function, which reads the input and interprets it to execute the correct program. Then the shell will open a pipe and send the command the location to the server. If it does not find any executable program that coincides with the input it will show an error message.

2.2 The server

The server will receive the message from the shell, and it will print the received data in the screen. depending on the command it should check some routines provided by *serverRoutines.c*. If one of those meets the requirements it will send a personalized acknowledgement message to the client, the shell. However, if none of the routines is fulfilled the server will send an *OK!* message, making the client know that everything went correctly.

2.2.1 Server routines

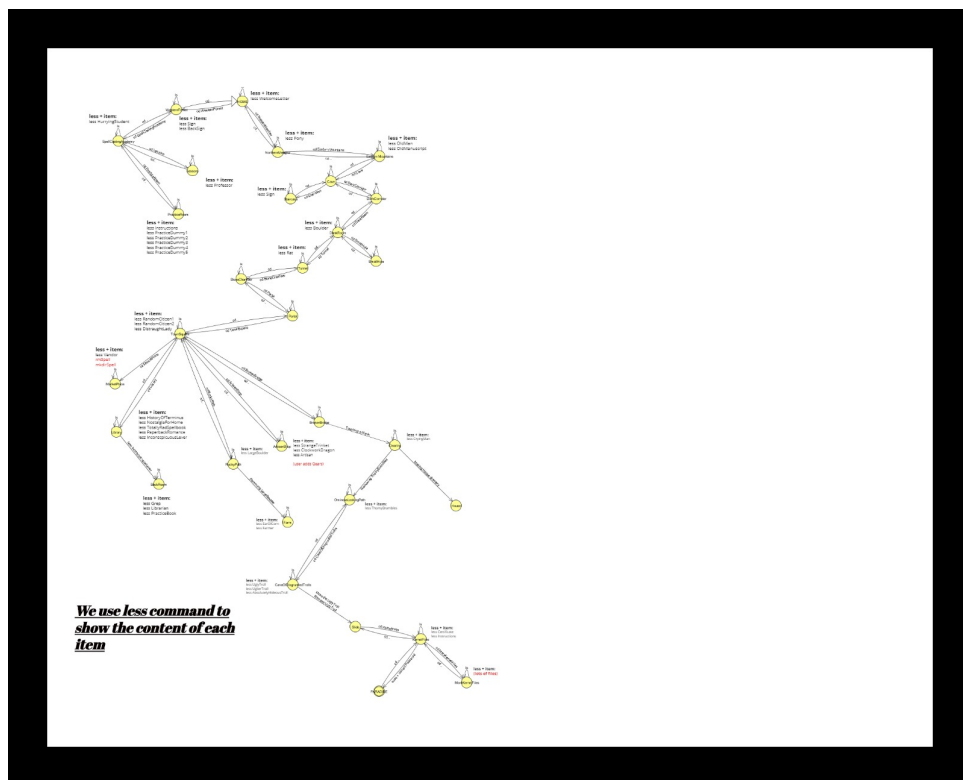
These routines have a very important role, depending on the command, location and argument, they will give permissions to different folders. The routines are divided in two parts, the first part is used to detect the command and location, the other part is called as a subroutine in the first one, it is used to send an acknowledgement message to the client.

2.3 Commands

The commands are simple c programs trying to raplicate the real commands in a less sophisticated way. The executable files of those programs are stored in the same directory and then called and executed by the main program. This are the commands we developed throughout this project:

ls, cd, cp, less, mv, rm, man, grep, sudo, ls, pwd and touch

2.4 The game



The automata shown above helps us to understand the structure of the game and the actions we can perform en each of the directories

3 Specification documents

In this sections we will show the specification document of each of the commands, reporting the name, description, authors, bugs and recommendations.

3.1 cd

NAME:

cd- This command changes the shell working directory and prints the name of the new path by the console.

SYNOPSIS:

cd [dir].

DESCRIPTION:

Changes the current directory to “dir”. The value of the home shell variable is the default directory. This command gets this directory and prints it. Then changes the directory to the input directory, obtaining this new one and printing its path.

SEE ALSO:

bash(1) , chdir() , getcwd()

REPORTING BUGS:

-If the permissions of the file prevent the change of the directory, the program will not be successfully executed.

-If the prefix of the input directory path is not a directory, the change of the directory could not be done.

-If the input directory does not exist.

AUTHOR:

Aitor Malo

3.2 cp

NAME:

cp - copy files and directories

SYNOPSIS:

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

DESCRIPTION:

Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY. SEE ALSO:

open() , write()

REPORTING BUGS:

- If the permissions of any directory (or both) are not enough, the copy of the directory will not be done.
- If any component of the input directories paths is are not a directory, command will not work successfully.
- If the first directory path does not exist.

AUTHOR:

Haizea Garaño

3.3 grep

NAME:

grep, egrep, fgrep, zgrep, zegrep, zfgrep - file pattern searcher

SYNOPSIS:

```
grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwXZ] [-A num] [-B num] [-C[num]]
[-e pattern] [-f file] [-binary-files=value] [-color[=when]]
[-colour[=when]] [-context[=num]] [-label] [-line-buffered]
[-null] [pattern] [file ...]
```


DESCRIPTION:

The `grep` utility searches any given input files, selecting lines that match one or more patterns. By default, a pattern matches an input line if the regular expression (RE) in the pattern matches the input line without its trailing newline. An empty expression matches every line. Each input line that matches at least one of the patterns is written to the standard output. SEE ALSO:

`ed(1)` , `ex(1)` , `gzip(1)` , `sed(1)`
, `re_format(7)`

REPORTING BUGS:

-If the directory does not exist, the `grep` command will not work.
-Also if the content of the directory is empty.

AUTHOR:

Unai Salas

3.4 less

NAME:

`less` - opposite of `more`

SYNOPSIS:

```
less [-[+ ]aABcCdeEfFgGiIJKLmMnNqQrRsSuUVwWX ]  
[-b space] [-h lines] [-j line] [-k keyfile]  
[-oO logfile] [-p pattern] [-P prompt] [-t tag]  
[-T tagsfile] [-x tab,...] [-y lines] [-[z] lines]  
[- shift] [+ [+ ]cmd] [-] [filename]...
```

(See the OPTIONS section for alternate option syntax with long option names.)

DESCRIPTION:

`Less` is a program similar to `more` (1), but which allows backward move- ment in the

file as well as forward movement. Also, less does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like vi (1). Less uses termcap (or terminfo on some systems), so it can run on a variety of terminals. There is even limited support for hardcopy terminals. (On a hardcopy terminal, lines which should be printed at the top of the screen are prefixed with a caret.)

Commands are based on both more and vi. Commands may be preceded by a decimal number, called N in the descriptions below. The number is used by some commands, as indicated.

SEE ALSO:

REPORTING BUGS:

-If the permissions to the file prevent the access to the file, the program will not work successfully.

-If any component of the input directory path is not a directory.

-If the input directory does not exist.

AUTHOR:

Haizea Garaño

3.5 man

NAME:

man - format and display the manual pages

SYNOPSIS:

man [-acdfFhkKtwW]

DESCRIPTION:

man formats and displays the manual pages. If you specify section, man only looks in that section of the manual. Name is normally the name of the manual page, which is typically the name of a command or function.

SEE ALSO:

`open()` , `read()`

REPORTING BUGS:

- If the name of the input command written does not exist, the `man` command will not be successfully executed and will not show the information of this command.

AUTHOR:

Unai Salas

3.6 `pwd`

NAME:

`pwd` - print name of current/working directory

SYNOPSIS:

`pwd`

DESCRIPTION:

Print the full filename of the current working directory.

SEE ALSO:

REPORTING BUGS:

-No reporting bugs.

AUTHOR:

Unai Salas

3.7 `touch`

NAME:

`touch` - change file timestamps

SYNOPSIS:

touch [OPTION]... FILE...

DESCRIPTION:

Update the access and modification times of each FILE to the current time. A FILE argument that does not exist is created empty, unless -c or -h is supplied. A FILE argument string of - is handled specially and causes touch to change the times of the file associated with standard output.

SEE ALSO:

readdir() , opendir()

REPORTING BUGS:

- No reporting bugs.

AUTHOR:

Aitor Malo

3.8 mv

NAME:

mv - move (rename) files

SYNOPSIS:

mv SOURCE... DIRECTORY

DESCRIPTION:

Rename SOURCE to DEST, or move SOURCE(s) to DIRECTORY.

SEE ALSO:

rename()

REPORTING BUGS:

-You have to specify the name of the destination file, otherwise it wouldn't work

AUTHOR:

Haizea Garaño

3.9 ls

NAME

ls - list directory contents

SYNOPSIS

ls [FILE]...

DESCRIPTION

List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

We differentiate the files and the directories by using the stat system call. If the fstats.st_mode and S_IFDIR is true that mean that it is a directory otherwise it is a file.

Mandatory arguments to long options are mandatory for short options too.

SEE ALSO

open

REPORTING BUGS:

- If the permissions to the file prevents the reading of it, this command will not be successfully executed.

- If the file or directory does not exist, the program will not work.

AUTHOR:

Unai Fernandez

3.10 sudo

NAME:

sudo - allows a permitted user to execute a command as another user.

SYNOPSIS:

```
sudo [password]
```

DESCRIPTION:

Allows a user to introduce a password. If the password is correct, the user will be able to change the directory and advance in the game. If the password is not correct, the game prevent the user to change the directory.

SEE ALSO:

```
strcmp()
```

REPORTING BUGS:

-No reporting bugs

AUTHOR:

Aitor Malo

WE SUPPOSE THAT ALL COMMANDS COULD CRASH BECAUSE OF INPUT/OUTPUT ERRORS, SUCH AS IF THE NUMBER OF THE ARGUMENTS DOES NOT MATCH WITH THE PROGRAM STRUCTURE.

4 Verification documents

4.1 Program or utility to verify: rm

Author(s) of the program: Unai Fernandez

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

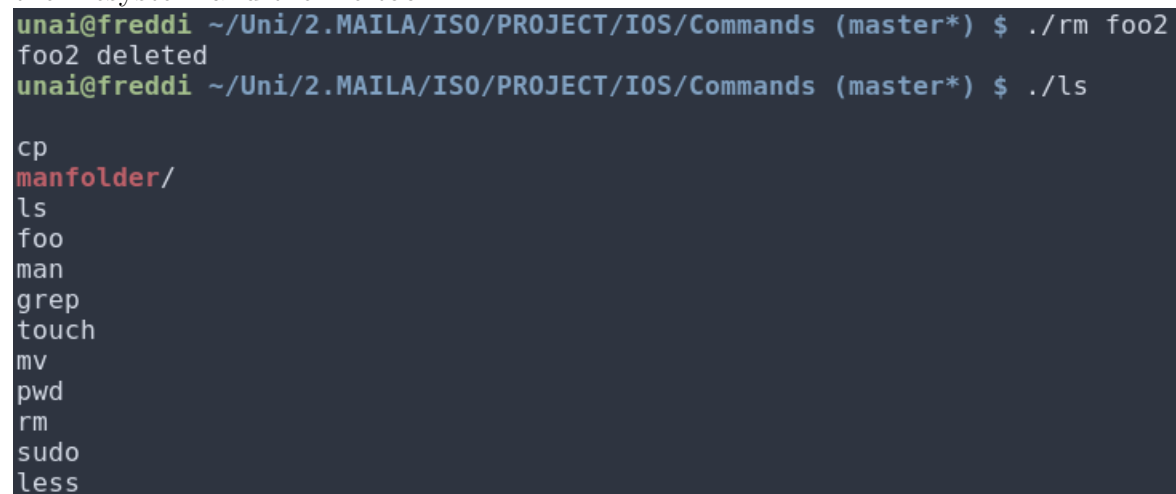
Objective: check rm command

Input: `./rm foo2`

Previewed output: "foo2 deleted"

Verification output: "foo2 deleted"

Explanation: It takes the file that we have passed by parameter and deletes a name from the filesystem and the file too.



```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./rm foo2
foo2 deleted
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./ls

cp
manfolder/
ls
foo
man
grep
touch
mv
pwd
rm
sudo
less
```

4.2 Program or utility to verify: ls

Author(s) of the program: Unai Fernandez

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check ls command

Input: ./ls

Previewed output: "cp manfolder/ ls man grep touch mv pwd rm sudo less "

Verification output: "cp manfolder/ ls man grep touch mv pwd rm sudo less "

Explanation: print the contents of the directory.

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master) $ ./ls
cp
manfolder/
ls
man
grep
touch
mv
pwd
rm
sudo
less
```

4.3 Program or utility to verify: less

Author(s) of the program: Haizea Garaño

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check less command

Input: ./less foo

Previewed output: "this is

a trial

bla bla bla bla "

Verification output: "this is

a trial

bla bla bla bla "

Explanation: print the contents of the file

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./less foo
this is
a trial
bla bla bla bla
```


4.4 Program or utility to verify: pwd

Author(s) of the program: Unai Salas

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check pwd command

Input: ./pwd

Previewed output: "You are located in /home/unai/Uni/2.MAILA/ISO/PROJECT/ISO/Commands
"

Verification output: "You are located in /home/unai/Uni/2.MAILA/ISO/PROJECT/ISO/Commands
"

Explanation: print name of current/working directory

A terminal window with a dark background. The prompt is 'unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) \$'. The command './pwd' has been entered, and the output is 'You are located in /home/unai/Uni/2.MAILA/ISO/PROJECT/IOS/Commands'.

4.5 Program or utility to verify: touch

Author(s) of the program: Aitor Malo

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check touch command

Input: ./touch foo

Previewed output: "(create the foo file) "

Verification output: "(create the foo file) "

Explanation: it creates the file with that name

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master) $ ./touch foo
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./ls

cp
manfolder/
ls
foo
man
grep
touch
mv
pwd
rm
sudo
less
```

4.6 Program or utility to verify: cp

Author(s) of the program: Haizea Garaño

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check cp command

Input: ./cp foo foo2

Previewed output: "cp manfolder/ ls foo man foo2 grep touch mv pwd rm sudo less"

Verification output: "cp manfolder/ ls foo man foo2 grep touch mv pwd rm sudo less"

Explanation: It copied the file foo and created another one with name foo2. The content is the same as we can see.

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./cp foo foo2
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./ls

cp
manfolder/
ls
foo
man
foo2
grep
touch
mv
pwd
rm
sudo
less

unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./less foo2
this is
a trial
bla bla bla bla
```

4.7 Program or utility to verify: mv

Author(s) of the program: Haizea Garaño

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check mv command

Input: `./mv foo manfolder/foo`

Previewed output: "You have moved from foo to manfolder/foo"

Verification output: "You have moved from foo to manfolder/foo"

Explanation: The file has been moved to the destination folder as indicated in the command

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./mv foo manfolder/foo
You have moved from foo to manfolder/foo
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./ls manfolder

cp
cd
ls
foo
man
grep
touch
pwd
less
```

4.8 Program or utility to verify: grep

Author(s) of the program: Unai Salas

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check grep command

Input: ./grep trial foo

Previewed output: "trial, line:2"

Verification output: "trial, line:2"

Explanation: It has searched the word that we have indicated in the command in the file that we have indicated

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./grep trial foo
trial, line:2
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./less foo
this is
a trial
bla bla bla bla
```

4.9 Program or utility to verify: man

Author(s) of the program: Unai Salas

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check man command

Input: `./man ls`

Previewed output: "(ls text entered in its man description)"

Verification output: "(ls text entered in its man description)"

Explanation: Look in the manfolder folder for the command file entered as parameter and print the content of that file

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./man ls
NAME
    ls - list directory contents

SYNOPSIS
    ls [FILE]...

DESCRIPTION
    List information about the FILEs (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor --sort is speci-
    fied.

    We differentiate the files and the directories by using the stat system call. If
    fstats.st_mode & S_IFDIR is true that mean that it is a directory otherwise it is a
    file.

    Mandatory arguments to long options are mandatory for short options
    too.

SEE ALSO
    open

REPORTING BUGS
```

4.10 Program or utility to verify: sudo

Author(s) of the program: Aitor Malo

Author(s) of the verification document: Unai Salas

Author(s) of the verification: Unai Fernandez

1st verification case

Objective: check sudo command

Input: `./sudo IHTFP`

Previewed output: "password is correct"

Verification output: "password is correct"

Explanation: We use the sudo command together with the password and the program

checks if the one entered as a parameter is equal to the password set

2nd verification case

Objective: check sudo command

Input: `./sudo skfla`

Previewed output: "Incorrect password"

Verification output: "Incorrect password"

Explanation: It does the same as in the first verification, but the password is now incorrect

```
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./sudo IHTFP
Password is correct
unai@freddi ~/Uni/2.MAILA/ISO/PROJECT/IOS/Commands (master*) $ ./sudo skfla
Incorrect password!
```

5 Theoretical concepts used in the project

5.1 System calls

A system call is the programmatic way in which a computer program requests a service from the kernel of the operating system on which it is executed. This may include hardware-related services, creation and execution of new processes, and communication with integral kernel services such as process scheduling. System calls provide an essential interface between a process and the operating system.

Several System calls have been used in this project, for example: `read()`, `open()`, `close()`, `write()`, ...

5.2 Commands

The main commands of this project are `cd`, `cp`, `grep`, `less`, `ls`, `man`, `mv`, `pwd`, `rm`, `sudo` and `touch`. We created a simple version of those and we used other commands such as `mkdir`, `echo` and `gcc` in the installer of the project.

5.3 Functions & Libraries

```
#include <stdio.h>
```

Meaning "standard input / output" heading, it is the heading in the C standard library that contains macro definitions, constants, and the declarations of functions and types used for the input of various standard and output operations.

Examples of used functions: `printf()`, `perror()`,

```
#include <unistd.h>
```

Library that contains functions for managing directories and files.

```
#include <string.h>
```

The `<string.h>` library contains a set of functions to manipulate strings: copy, change characters, compare strings, etc.

Examples of used functions: `strcat()`, `strcmp()`.

```
#include <fcntl.h>
```

Defines the flags for the opening modes of a file.

```
#include <errno.h>
```

Declares a list of error constants returned by the input and output functions.

```
#include <sys/stat.h>
```

The `<sys / stat.h>` header defines the structure of the data returned by the functions `fstat ()`, `lstat ()`, and `stat ()`.

```
#include <dirent.h>
```

5.4 Pipes

A pipeline is a mechanism for inter-process communication using message passing. A pipeline is a set of processes chained together by their standard streams, so that the output text of each process (`stdout`) is passed directly as input (`stdin`) to the next one. In this project pipes are used to connect processes between the server and the shell.

5.5 Permissions

In order to lock some of the directories in the project, we used permissions. In the installer of the game we used,

```
chmod 444 [NAME OF THE FOLDER]
```

to give only reading permissions to some of the directories. That means that you can see the content of the folder but you can't access it. When the player types specific commands in specific locations, the folders are unlocked. Said in other words, the server gives the folder all permissions, by using,

```
chmod(path, S_IRWXU);
```

function.

5.6 Processes

A process is basically an executing program. Multiprocessing refers to the management or execution of many processes at the same time. We have used this technique in the shell, using the `fork()` system call, that creates a child process for the process in which we call this function. This way we can identify the different processes and the communication between them.

6 Distribution

6.1 Unai Fernandez

Completed tasks: ls command, rm command, server, installer

6.2 Haizea Garaño

Completed tasks: mv command, cp command, less command

6.3 Aitor Malo

Completed tasks: cd command, sudo command, touch command

6.4 Unai Salas

Completed tasks: pwd command, man command, grep command

That was the main organization of the project, even if all the members of the group has a command in charge, every team mate collaborated in all the tasks in the project, and we are all aware of the work made by the other members. Every week a group meeting was scheduled to organize the work.

7 Installation Guide

This section is a guide for installing Terminus. Before start with the installation, the *Terminus.zip* file should be downloaded from our github repository [1]. Having done that, the next step is to unzip the compressed file to whatever folder you want. Once the folder inside has been extracted to the chosen directory, the *install.sh* file should be executed. If you haven't permissions to execute the file, you have to give them by typing:

```
chmod +x install.sh
```

However if all is going correctly the game folders will be created as well as the executable files for the server and the shell.

7.1 After the installation

After the installation the installer will ask you if you want to play or not. If the answer is n (no) it will simply exit the program. However, if the answer is y (yes) it will execute the shell. But be careful, you need to execute the server at the same time. To do this you have to open another terminal and type:

```
cd [path to the game]
```

Once in the game folder, type

```
./server
```

to execute the server. Having done those things successfully you are now able to play our game.

8 Manual

You are a newcomer to this land and, since you have reason, you remember stories about the paradise that is hidden there. You have always wanted to come and, finally, you have succeeded. Congratulations!

But now you must use your skills (the commands) to enter this land, where ancient magic and unexplored secrets await you. Explore every corner of this land, learn from it and with it and reach paradise.

Remember that the trip is to be enjoyed while you learn.

Good luck in your search and we hope you enjoy the journey.

9 Conclusions

Through this project we have managed to deepen in a practical way the theoretical knowledge acquired in class. We have learned to use functions of libraries, system calls, pipes, etc while retaking contact with the c language in a more complete and practical way.

We have also been able to work on the permissions on folders and files in a very interesting way and we are happy with the project, being the replacement of the more conventional projects.

We believe that we have acquired the necessary knowledge in a practical way through this project in the same way as if we had done it in a conventional way.

Bibliography

- [1] S. Unai, M. Aitor, G. Haizea, and F. Unai. Ios: Terminus. [Online]. Available: <https://github.com/windymeck/IOS>