(Group 3) # Report II (03/04/2020)

SUMMARY

Summary of the changes we made from the last week (24/03/2020):

- Correct the errors (the feedback).
- Add in the report the verification document of some commands
- After creating the script to make the world, we put permissions to some files and folders, but not to all of them.
- When we leave the game, the directories and files that we created in the script it deletes again.

LIST OF COMMANDS

rm

Author: Iñigo

Specification

The command 'rm' of our game is a command that removes an item of a specific place. For example, if we are in a long and narrow path and there is a rock in the middle, we should remove it to follow with the travel.

This aim of the command in Bash is to remove a file and if we want to remove a directory we should use the command rmdir, if we if we remove a directory with 'rm' we will get the following error:

```
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/Desktop$ rm dir
rm: cannot remove 'dir': Is a directory
```

The system call we used to program the command is “unlink()”. unlink() deletes a name from the file system. If that name was the last link to a file and no processes have the file open the file is deleted and the space it was using is made available for reuse.

If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed. On success, zero is returned. Nevertheless, on error, -1 is returned,

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

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

  if(argc!=2){
    printf("Please, introduce the correct parameters: ./rm.o nameOfFile\n");
  }
  else{
    int exist;
    exist=unlink(argv[1]);
    if(exist==0){
      printf("File deleted\n");
    }
    else{
      perror("unlink");
    }
  }

  return 0;
}
```

Verification

1st verification case

Objective: execute the command putting anything else.

Input: $ ./rmFile

Previewed output: Error, put the correct parameters.

Verification output:

```
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/git/IOSWork/Terminus_project/commands$ ./rm.o
Please, introduce the correct parameters: ./rm.o nameOfFile
```

Explanation: If we don't put any name of a file after de command, the command doesn't know what file we want to delete, and because of that situation, it shows the error.

2nd verification case

Objective: try to remove a file that does not exist.

Input: $ ./rmFile inventedName

Previewed output: Error

Verification output:

```
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/git/IOSWork/Terminus_project/commands$ ./rm.o file1
unlink: No such file or directory
```

Explanation: If we put a non existing file in the command line, the program doesn't going to know what file is the one that has to delete.

3rd verification case
Objective: Execute the program correctly, and wait for the results. Input: $ ./rmFile file Previewed output: File deleted Verification output:    *Explanation:* When we execute the command, the program calls to the system call *unlink()*, who returns 0. The return of the Integer 0 means that the system call Works properly. However, if the system call returns -1, it means that there is an error.

mv

Author: Iñigo

Specification

The command 'mv' of our game is a command that moves one item from one place into another place . For example, if we are in a long and narrow path and there is a rock in the middle, we should remove it to follow with the travel.

This aim of the command in Bash is to rename a file. If we want to rename the file "house" and put the name "home" we have to write the command like this:

```
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/git/IOSWork/Terminus_project/mv/prueba2$ ls
house
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/git/IOSWork/Terminus_project/mv/prueba2$ mv house home
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/git/IOSWork/Terminus_project/mv/prueba2$ ls
home
```

The system call we used to program the command is “rename(char)”. The rename() function shall change the name of a file. The old argument points to the pathname of the file to be renamed. The new argument points to the new pathname of the file.

If either the old or new argument names a symbolic link, rename() shall operate on the symbolic link itself, and shall not resolve the last component of the argument. If the old argument and the new argument resolve to the same existing file, rename() shall return successfully and perform no other action.

Upon successful completion, rename() shall return 0; otherwise, -1 shall be returned.

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

void main (int argc, char **argv){

if(argc!=3){
printf("Please, introduce the correct parameters: ./mvFile.o oldName newName \n");
}
else{
        int w= rename(argv[1], argv[2]);
        if(w==0){
                printf("The %s is moved to %s\n",argv[1], argv[2]);
        }
        else{
                perror("The following error ocurred: ");
        }
}

}
```

Verification

1st verification case

Objective: Execute the program correctly, and wait for the results.

Input: $ ./mvFile oldName newName

Previewed output: oldName file moved to newName

Verification output:

```
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/Desktop$ ./mvFile.o file2 file3
The file2 is moved to file3
```

Explanation: When we execute the command, the program calls to the system call *rename()*, who returns 0. The return of the Integer 0 means that the system call Works properly. However, if the system call returns -1, it means that there is an error.

2nd verification case

Objective: execute the command putting anything else.

Input: $ ./mvFile

Previewed output: Error, put the correct parameters.

Verification output:

```
inigopm@LAPTOP-HAH37RJI:/mnt/c/Users/usuario/Desktop$ ./mvFile.o file
Please, introduce the correct parameters: ./mvFile.o oldName newName
```

Explanation: If we don't put any name of a file after de command, the command doesn't know what file we want to move, and because of that situation, it shows the error.

3rd verification case
Objective: try to move a file that does not exist. Input: $ ./mvFile inventedName newName Previewed output: Error Verification output:    Explanation: If we put a non existing file in the command line, the program doesn't going to know what file is the one that has to move.

ls

Author: Marcos Elosegui

Specification

The "ls" command is used to see the paths and items available in the place that you are in the game. In order to implement this command we've used several system calls like:

- getcwd(): copies an absolute pathname of the current working directory to the array pointed to.
- opendir(): open a directory.
- readdir(): read a directory.
- closedir(): close a directory.

```c
int main(int argc, char* argv[])
{
    char cwd[PATH_MAX];
    DIR *dir;
    struct dirent *ent;

    if( argc == 1)
    {
        getcwd(cwd, sizeof(cwd));
        if ((dir = opendir (cwd)) != NULL) {
        /* print all the files and directories within directory */
            while ((ent = readdir (dir)) != NULL) {
                printf ("%s\n", ent->d_name);
            }
        closedir (dir);
        } else {
        /* could not open directory */
        perror ("");
        return EXIT_FAILURE;
        }
    } else {
        perror ("");
        return EXIT_FAILURE;
    }
}
```

1st verification case
Objective: execute the command. Input: $ ./ls Previewed output: Paths and files of the current location. Verification output: *Explanation:* The command will show us the paths and files of the current location.

```
myShell0> ls
WelcomeLetter.txt
NorthernMeadow
.DS_Store
.
..
WesternForest
myShell0>
```

2nd verification case
Objective: execute the command with a path. Input: $ ./ls path Previewed output: No such file or directory. Verification output:  *Explanation:* If we put a path the ls command won't show us the paths and the files of that path like in the game.

pwd

Author: Ander Barrio Campos

What is the pwd command in UNIX?

The pwd command is a command line utility for printing the current working directory. It will print the full system path of the current working directory to standard output. By default the pwd command ignores symlinks, although the full physical path of a current directory can be shown with an option. The pwd command is normally a shell builtin; meaning it is part of the code that runs the shell rather than an external executable.

How to print the current working directory

To print the current working directory run the pwd command. The full path of the current working directory will be printed to standard output.

PWD implementation in C:

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

int main() {
    char cwd[1024];
    chdir("/path/to/change/directory/to");
    getcwd(cwd, sizeof(cwd));
    printf("Current working dir: %s\n", cwd);
}
```

As we notice, the use of the system call *getcwd()* is primordial to implement the code of pwd.

Description:

The getcwd() function copies an absolute pathname of the current working directory to the array pointed to by cwd, which is of length size.

If the current absolute path name would require a buffer longer than size elements, -1 is returned, and errno is set to ERANGE; an application should check for this error, and allocate a larger buffer if necessary.

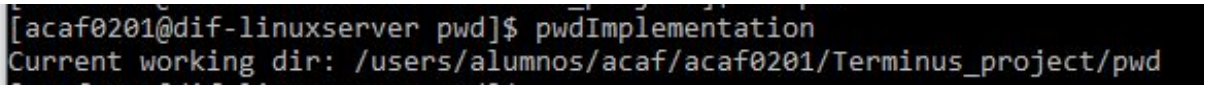
If cwd is NULL, the behaviour of getcwd() is undefined.

Return Value

-1 on failure (for example, if the current directory is not readable), with errno set accordingly, and the number of characters stored in cwd on success. The contents of the array pointed to by cwd is undefined on error.

Note that this return value differs from the getcwd(3) library function, which returns NULL on failure and the address of cwd on success.

After compiling and executing the code in C, we check that it works properly:



1st verification case
Objective: *check the correct functioning of the command pwd.* Input: *./pwd* Previewed output: It should print the current working directory. Allocated at the end of the path. Verification output: Current working dir: /users/alumnos/acaf/acaf0201/Terminus_project/commands *Explanation: As we can see, it works as we wanted to work*, using the system call getcwd().

2nd verification case
Objective: *check if the user writes the command pwd + whatever, it continues working correctly; as the original command of Linux.* Input: *./pwd testing* Previewed output: It should print the current working directory. Allocated at the end of the path. Ignoring the "testing" word. Verification output: Current working dir: /users/alumnos/acaf/acaf0201/Terminus_project/commands Explanation: The implementation just focuses on the pwd, instead of other words, using the system call getcwd().

CAT

AUTHOR: Julen Urroz

NAME: cat - concatenate

SYNOPSIS: cat [File]...

DESCRIPTION:

Cat command concatenates FILE(s) to standard output.. We are using cat as a substitute of the “less” command in the original terminus game, since it’s functionality really was the one of cat in the first place. Players are able to interact with objects/items in the game with this command, which will show the description of said item.

SEE ALSO:
System calls used in this command:

- int open(const char **pathname*, int *flags*);
- ssize_t read(int *fd*, void **buf*, size_t *count*);
- ssize_t write(int *fd*, const void **buf*, size_t *count*);
- int close(int *fd*);

```c
#include <fcntl.h>
#include<sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include<dirent.h>
#include<error.h>
#include <stdlib.h>


int main(int argc , char *argv[])
{
  ssize_t byteread;
  char* buffer;
  fflush(stdout);
  int pagesize = getpagesize();
  int fd;
  int i;

  for(i=1; i<argc; i++){
     fd = open(argv[i], O_RDONLY);
     if(fd < 0){
        perror("The following error ocurred");
        return -1;
     }
        write(1,buffer,byteread);
     }
     if(byteread < 0){
        perror("The following error ocurred");
        return -1;
     }
     close(fd);
  }
  return 0;
} 0;

```

VERIFICATION:

1st verification case
Objective: Test program behaviour with a correct pathname. Input: cat StaircaseSign.txt Previewed output: *DEAD END* *(its what the file contains)* Verification output: *DEAD END* *Explanation:* The very basic functionality of the cat command is to print the contents of a file to standard output. The program completes with no problems and the correct output is obtained.

2st verification case

Objective: Test program behaviour with a more than one pathnames.

Input: cat Sign.txt BackSign.txt (both valid pathnames)

Previewed output:

Spell Casting Academy: The Elite School of Magic Today Only: Free Introductory Lessons!

Novices welcome!

If you ever want to go directly Home, just type 'cd ~' or just plain old `cd' and you'll come back Home. Getting back might be more difficult though.

Verification output:

Spell Casting Academy: The Elite School of Magic Today Only: Free Introductory Lessons!

Novices welcome!

If you ever want to go directly Home, just type 'cd ~' or just plain old `cd' and you'll comeback Home. Getting back might be more difficult though.

Explanation:

Now we test the real functionality (and source of the name) of the command. The contents of the files are concatenated. This means they are displayed in standard output in the same order they were given as inputs, one after the other.

3rd verification case

Objective: Test program behaviour with a non existing file input.

Input: cat aaa (aaa being a false pathname)

Previewed output:

"The following error occurred: no such file or directory"

Verification output:

"*The following error occurred: no such file or directory*"

Explanation:

The program uses the open() system call at the start, and checks if the returned file descriptor is negative in order to know if any error happened while opening the file.

If it is negative a message is printed using the perror function with the format "The following error happened: *whatever error"*.

4th verification case

Objective: Test with empty input

Input: cat (no pathname given)

Previewed output:

No visible output

Verification output:

No visible output

Explanation:

The program loops from 1 to argc-1 to read files 1 by 1. In this case argc=1 so it never

enters the loop and the program just finishes without printing any output.

GREP

AUTHOR: Julen Urroz

NAME: grep - global regular expression print

SYNOPSIS: grep Pattern [File]...

DESCRIPTION:

Grep command searches the input pattern in the designated file. Lines where the pattern is found are printed with the position of the pattern highlighted.

SEE ALSO:

System calls used in this command:

- int open(const char **pathname*, int *flags*);
- ssize_t read(int *fd*, void **buf*, size_t *count*);
- int stat(const char **pathname*, struct stat **statbuf*);

- ssize_t write(int *fd*, const void **buf*, size_t *count*);
- int close(int *fd*);

VERIFICATION:

1st verification case
Objective: Test basic functionality Input: grep east Pony.txt Previewed output: and knocks you off. He then looks towards the east as if suggesting that you head in that direction. Verification output: and knocks you off. He then looks towards the east as if suggesting that you head in that direction. *Explanation:* The command inputs are valid ones so the execution goes as expected. There is only one instance of the pattern “east” in this file so the line with it gets printed with the pattern highlighted.

2nd verification case

Objective: Test zero argument case

Input: grep

Previewed output:

No output at all will be printed and the program finish without errors.

Verification output:

No output is obtained, no error messages ,and the shell prompt is displayed again.

Explanation:

The current version of the program checks if argc equals 3 at the start and only does

anything else if this is the case.

3rd verification case

Objective: Test behaviour of program with an invalid pathname as input

Input: grep east aaa

Previewed output:

Error message displayed on standard ouput: "no such file or directory."

Verification output:

"stat(): No such file or directory"

Explanation:

Right after checking the argc value, a call to stat() is performed to check if the file

with the given pathname exists. If it doesn't(if stat doesn't finish with normality and return 0) a message is printed with perror.

```c
void print_colored(char line[], char* pattern)
{
   char* head = line;
   char* color_start = strstr(line, pattern);


   if(color_start != NULL){
      char* color_end = color_start + strlen(pattern);

      while(head != color_start){
         write(0,head,sizeof(char));
         head++;
      }
      system("echo -en  \x1b[31m");
      while(head !=color_end){
         write(0,head,sizeof(char));
         head++;
      }
      system("echo -en \x1b[0m");
      while(*head != '\0'){
        write(0,head,sizeof(char));
        head++;
      }
      write(0,"\n",1);
   }
}


void match_pattern(char* argv[])
{
   int fd,i=0,r;
   char buf;
   char line[256];

   if((fd = open(argv[2],O_RDONLY)) < 0){
       perror("The followirng error happened: ");
       exit(1);
   }
   while((r = read(fd, &buf, sizeof(char))) != 0){
      if(buf == '\n'){
         line[i]='\0';
         print_colored(line,argv[1]);
         memset(line,0,sizeof(line));
         i=0;
      }
      else{
         line[i++]=buf;
      }
   }
}


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

   struct stat stt;
   if(argc==3)
   {
       if(stat(argv[2],&stt)==0) //check if the file exists
          match_pattern(argv);
       else
       {
           perror("stat()");
           exit(1);
       }
   }
   return 0;
}
```

CD

AUTHOR: Marion BERNARD

SPECIFICATION:

NAME: cd - change the working directory

SYNOPSIS: cd [-L | -P] [*directory*]

cd -

DESCRIPTION:

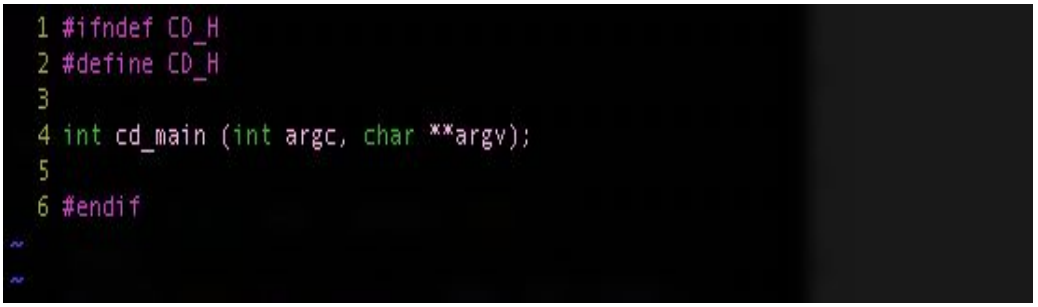
The cd command is a command-line shell used to change the current working directory. There are two possible values that can be returned : 0 if the system call succeeds in changing current directory or -1 if it fails.

SEE ALSO:

System calls used in this command:

- **int chdir(const char** **path****);**
- **char *getenv(const char *****name****);**

cd.h



cd.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <limits.h>

#include "cd.h"

int cd_home(void)
{
    const char *const home = getenv("HOME");
    if (home){
        printf("chdir(%s) = %d\n", home, chdir(home));
        return chdir(home);
    }
    return -1;
}

int cd_main (int argc, char **argv)
{
    switch (argc) {
    case 1:
        cd_home();
        break;
    case 2:
        if (!strcmp(argv[1], "~"))
            cd_home();
        else
            printf("chdir(%s) = %d\n", argv[1], chdir(argv[1]));
        break;
    default:
        fprintf(stderr, "usage: cd <directory>\n");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}
```

Makefile

```
#Makefile

all: main

main: main.o cd.o
	gcc main.o cd.o -Wall -O

print_page.o: print_page.c
	gcc -o cd.o -c cd.c -Wall -O

main.o: main.c cd.h
	gcc -o main.o -c main.c -Wall -O

clean:
	rm -rf *.o core

mrproper: clean
	rm -rf main
```

VERIFICATION:

1st verification case
Objective: Test basic functionality Input: cd NorthernMeadow Previewed output: NorthernMeadow is a knowing folder. The program is supposed to change the current directory. that direction. Verification output: The program just changed its current directory for the user. No error message and return EXIT_SUCCESS. Explanation: By using the chdir() system call, the program is able to know if a directory does exists or not. If it does, then the current root is changed.

2nd verification case

Objective: Test zero argument case or "~" in second argument

Input: cd / cd ~

Previewed output:

This particular case will take you back to your default directory of the user called "root".

Verification output:

No error messages, and the shell prompt is displayed again. This message appears : "You are at the first room".

Explanation:

The program check the number of arguments and if there is more than one, it checks if the second one is similar to "~". In this two cases, it will use an auxiliary function "cd_home" to go back to the user root.

3rd verification case

Objective: Test behaviour of program with an invalid folder name as input

Input: cd test

Previewed output:

This wrong input is supposed to write in the stdout an error message. In deed, the folder test does not exist.

Verification output:

Error message displayed on standard output: "No such file or directory."

Explanation:

If the program does not know the folder name in argv[1] by using the chdir() system call. Then, the folder is unknown and the cd command cannot be applied.

touch

Author: Marcos Elosegui

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

#define DATA_SIZE 1000

int main()
{
    char data[DATA_SIZE];

    FILE * fPtr;

    fPtr = fopen("argv[1]", "w");


    if(fPtr == NULL)
    {
        printf("Unable to create file.\n");
        exit(EXIT_FAILURE);
    }

    printf("Enter contents to store in file : \n");
    fgets(data, DATA_SIZE, stdin);

    fputs(data, fPtr);

    fclose(fPtr);

    printf("File created and saved successfully. :) \n");

    return 0;
}
```

touch: create a file.

fopen(): open a file, if doesn't exist create it.

fgets(): fetch the data from the user by the Standard Input.
fputs(): write the fetched data in a file.
fclose(): close a file.

man

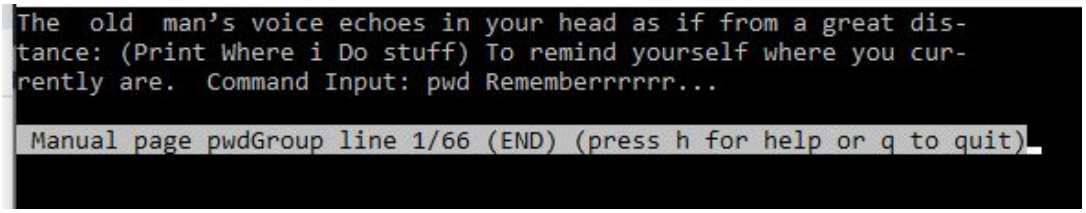
Author: Ander Barrio Campos

man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.

Every manual is divided into the following sections:

- Executable programs or shell commands
- System calls (functions provided by the kernel)
- Library calls (functions within program libraries
- Games
- Special files (usually found in /dev)
- File formats and conventions eg /etc/passwd
- Miscellaneous (including macro packages and conventions), e.g. groff(7)
- System administration commands (usually only for root)
- Kernel routines [Non standard]

In the game, we have created our customized man pages based on the Terminus Game. We have added the explanation of the commands cd, grep, ls, mv, touch, cp, man and pwd. Here is an example of the command “man pwd”:



1st verification case
Objective: *check the correct functioning of the man pages created.* Input: *man ./manGroup* Previewed output: It should show us the man page created. Verification output: I'm the old man dangit! You can't try to get more information about me. Here are all the commands you can man: cd, ls, rm, mv, exit, help, man, touch, grep, pwd. *Explanation: As we can see, it works as we wanted to work.*

2nd verification case
Objective: *check if the user writes an unexisting man page, the system must notify that it does not exist.* Input: *man ./manGroup2* Previewed output: A warning message. Verification output: man: ./manGroup2: No such file or directory No manual entry for ./manGroup2 Explanation: It does not work because it does not exist manGroup2 page.

3rd verification case
Objective: *check if the user writes an extra word, that it continue working.* Input: *man ./manGroup 3* Previewed output: A warning message Verification output: No manual entry for 3 (Alternatively, what manual page do you want from section 3?) Explanation: As we just have created a single man page for every command, we have set it as the default page. That is why it appears although whatever "thing" is

added.

CP

What is the cp command?

The cp command is a command-line utility for copying files and directories. It supports moving one or more files or folders with options for taking backups and preserving attributes. Copies of files are independent of the original file unlike the mv command.

How to copy a file

To copy a file with the cp command pass the name of the file to be copied and then the destination. In the following example the file foo.txt is copied to a new file called bar.txt. The cp command will also create the new file as part of the operation.

- ls
- foo.txt
- cp foo.txt bar.txt
- ls
- foo.txt bar.txt

How to copy multiple files

To copy multiple files using the cp command pass the names of files followed by the destination directory to the cp command.

How to copy a directory

By default the cp command will not copy directories. Attempting to copy a directory results in an error.

- cp directory/ foo

- cp: omitting directory 'directory/'

To copy a directory pass the -R flag. This will recursively copy a folder and create a copy.

PLANNING (next steps)

- Include the permissions when creating the directory tree of the game that will restrict some areas to the player until he/she meets some requirements.
- Write the code that will alter those permissions when the player meets the objective.