Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Manejo e implementación de archivos
Segundo Semestre 2021
Catedrático: Ing. Oscar Paz
Aux: Mario Obed Morales

PROYECTO 1

Nombre	Carnet
Maxwellt Joel Ramírez Ramazzini	201709328

CLASE ENTRADA

```
void entrada::MenuInicio(){
    cout << "------------------------------BIENVENIDO A LA APLICACIÓN DE COMANDOS\n" << endl;
    cout << "------------------------------ESCRIBA EXIT PARA SALIR DE LA APP\n" << endl;
    cout << "Escriba acá su comando:    ";

    while (true){
        //Recibe comando
        string comando;
        getline( &: cin,  &: comando);
        //Limpiamos la pantalla
        LimpiarPantalla();
        //string AuxCom = Minuscula(comando);
        //Validamos si es o no exit, para terminar o no el programa
        if (Equals( Uno: comando,  Dos: "exit"))
        {
            exit( status: -1);
        }
        //Validamos el comando y buscamos los tokens
         tok= busquedaToken(comando);

        //Borramos el token que acaba de registrar
        comando.erase( pos: 0, n: tok.length()+1);

        //Enviamos el resto de la cadena para que se hagan los splits
        tokRet = splitTok( entrada: comando);
        //Probaremos si funcionan los scripts
        comandos( comando: tok, tokens: tokRet);

        cout << "\n**********Programa en pausa**********\nPresione enter, para poder continuar:" << endl;
        getline( &: cin, &: comando);
        LimpiarPantalla();
        cout << "------------------------------BIENVENIDO A LA APLICACIÓN DE COMANDOS\n" << endl;
        cout << "------------------------------ESCRIBA EXIT PARA SALIR DE LA APP\n" << endl;
        cout << "Escriba acá su comando:    ";
    }
}
```

En esta clase se manejan splits, menús y analizadores para poder obtener los comandos, junto a cada uno de sus parámetros.

CLASE LOGDISCOS

```
void LogDiscos::ComandoMkdisk(vector<string> parametros){
    string Auxpara;
    string tamanio;
    fit ="";
    unit  ="";
    Dir = "";

    for (string parametro: parametros) {
        Auxpara = parametro.substr( pos: 0, n: parametro.find( s: ":"));
        parametro.erase( pos: 0, n: Auxpara.length()+2);
        Auxpara = Auxpara.substr( pos: 0, n: Auxpara.find( s: "-"));

        if (entradita.Equals( Uno: Auxpara, Dos: "fit")){
            if (fit.empty()){
                fit = parametro;
            }else{
                entradita.AlertaError( comando: "ERROR EN  MKDISK", alerta: "Sucedio un problema con el fit \"F\" en la linea:"+ Auxpara);
            }
        }else if (entradita.Equals( Uno: Auxpara, Dos: "unit")){
            if (unit.empty()){
                unit = parametro;
            }else{
                entradita.AlertaError( comando: "ERROR EN  MKDISK", alerta: "Sucedio un problema con la unit \"U\" en la linea:"+ Auxpara);
            }
        }else if (entradita.Equals( Uno: Auxpara, Dos: "size")){
            if (tamanio.empty()){
                tamanio = parametro;
            }else{
                entradita.AlertaError( comando: "ERROR EN  MKDISK", alerta: "Sucedio un problema con el tamanio \"SIZE\" en la linea:"+ Auxpara);
            }
        }else if (entradita.Equals( Uno: Auxpara, Dos: "path")){
            if (Dir.empty()){
                Dir = parametro;
                Dir = BorrarEspacio( var: Dir);
            }else{
                entradita.AlertaError( comando: "ERROR EN  MKDISK", alerta: "Sucedio un problema con la direccion \"PATH\" en la linea:"+ Auxpara);
            }
        }else{
            entradita.AlertaError( comando: "ERROR EN MKDISK", alerta: "Se encontro un parametro indefinido o que no se esperaba, en la linea: "+ Auxpara);
```

En esta clase trabajamos distintos métodos para la lógica del Disco, donde los creamos, eliminamos y también modificamos con cada una de las particiones.

CLASE DECISIÓN

```cpp
#include "Decision.h"
#include "cstring"
#include "iostream"
#include "stdlib.h"
#include "./Entrada.h"
using namespace std;

entrada entra;

decision::decision(){

};
bool decision::Respuesta(string mensaje){
    std::cout << mensaje << "Escriba \"y\" para confirmar, si escribe cualquier otra letra sera tomada como cancelar\n";
    string respuesta;
    getline( &: cin, &: respuesta);
    if (entra.Equals( Uno: respuesta,  Dos: "y"))
    {
        return true;
    }
    return false;

}

void decision::Mensaje(string comando, string mensaje){
    cout << "\033[0;42m(" + comando + ")~~> \033[0m"<< mensaje << endl;
}
```

En esta clase manejamos algunas respuestas para cuando se efectué algún comando, ya se correcto o también advertencias.

METODO RMDISK

```
string LogDiscos::CorrigeParametrosFit(string fit) {
    if (fit.empty())
    {
        fit = "BF";
    }
    return fit;
}
string LogDiscos::CorrigeParametrosUnit(string unit){
    if (unit.empty())
    {
        unit = "M";
    }
    return unit;
}

void LogDiscos::ComandoRmdisk(vector<string> entrada){
    string dir;
    string AuxTokFor;
    //Verificamos que no este vacia la cadena
    if (entrada.size() ==0){
        entradita.AlertaError( comando: "COMANDO RMDISK",  alerta: "No se encontro ninguna dirección valida, vuelva a intentarlo");

    }
    //Revisamos la dirección y eliminamos los parametros ~:~ para poder usarla
    for (string tokAux: entrada) {
        AuxTokFor = tokAux.substr( pos: 0, n: tokAux.find( s: ":"));
        tokAux.erase( pos: 0, n: AuxTokFor.length()+2);
        AuxTokFor = AuxTokFor.substr( pos: 0, n: AuxTokFor.find( s: "~"));
        if (entradita.Equals( Uno: AuxTokFor,  Dos: "path")){
            //Si es correcto, copiamos el valor hacia la variable dir
            dir = tokAux;
        }else{
            //Error en la direccion
            dir="";
            entradita.AlertaError( comando: "COMANDO RMDISK", alerta: "Se encontro un error en la direccion, en la linea: "+ AuxTokFor);
            break;
```

Comando RMDISK, se encuentra dentro de la clase LogDiscos para ejecutar sus funciones y que sea mucho más fácil el manejo de cada una de estas y los métodos no tengan oportunidad a perder datos.

CLASE MOUNT

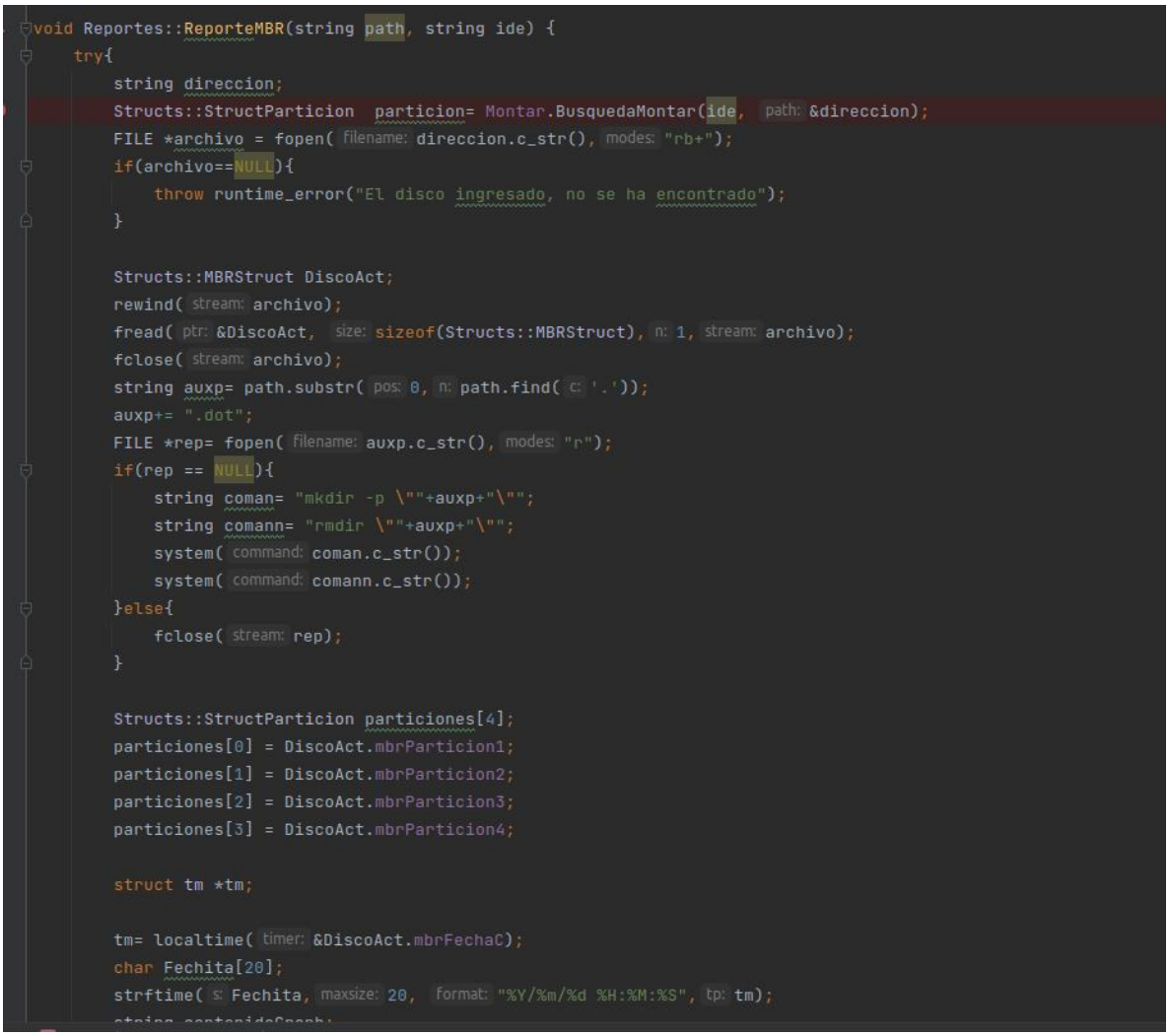
```
void Mount::Desmontar(vector<string> entrada) {
    vector<string> datos = {"id"};
    string ide;

    for (int i = 0; i < entrada.size(); i++) {
        string posi = entrada.at( n: i);
        string idAux = posi.substr( pos: 0,  n: posi.find( s: "~"));
        posi.erase( pos: 0, n: idAux.length()+3);

        if(auxx.Equals( primero: idAux, segundo: "id")){
            auto itr :iterator<...> = find( first: datos.begin(), last: datos.end(), val: idAux);
            datos.erase( position: itr);
            ide = posi;
        }
    }
    if(datos.size()!=0){
        auxx.Alerta( comando: "COMANDO UNMOUNT",  accion: "Ocurrio un error, revise todos los"
                                        "parametros ingresados");
        return;
    }
    DesmontarAgain(ide);
}

void Mount::DesmontarAgain(string ide) {
    try{
        string p=ide;
        if(!(ide[0]=='v' && ide[1] == 'd')){
            throw runtime_error("Identificador no valido o no existe");
        }
        char letra=ide[ide.length()-2];
        ide.erase( pos: 0, n: 2);
```

En esta clase montamos y desmontamos particiones dentro de los discos, manejamos todos los identificadores que estas usan y conexiones hacia los discos.

REPORTES

```
void Reportes::generaReporte(vector<string> entradita, Mount monta) {
    Montar = monta;

    vector<string> parametros = {"path", "name","id"};
    string path,id, name;

    for (string posi: entradita) {
        string Auxid = auxMet.Minus( entrada: posi.substr( pos: 0, n: posi.find( c: ':')));
        posi.erase( pos: 0, n: Auxid.length()+2);
        Auxid = Auxid.substr( pos: 0, n: Auxid.find( s: "~"));
        if(posi.substr( pos: 0, n: 1)=="\""){
            posi = posi.substr( pos: 1, n: posi.length()-2);
        }
        if(auxMet.Equals( primero: Auxid, segundo: "name")){
            if(count( first: parametros.begin(), last: parametros.end(), value: Auxid)) {
                auto itr :iterator<...> = find( first: parametros.begin(), last: parametros.end(), val: Auxid);
                parametros.erase( position: itr);
                name = posi;
            }
        }else if(auxMet.Equals( primero: Auxid, segundo: "id")){
            if(count( first: parametros.begin(), last: parametros.end(), value: Auxid)){
                auto itr :iterator<...> = find( first: parametros.begin(), last: parametros.end(), val: Auxid);
                parametros.erase( position: itr);
                id = posi;
            }
        }else if(auxMet.Equals( primero: Auxid, segundo: "path")){
            if(count( first: parametros.begin(), last: parametros.end(), value: Auxid)){
                auto itr :iterator<...> = find( first: parametros.begin(), last: parametros.end(), val: Auxid);
                parametros.erase( position: itr);
                path = posi;
            }
        }
    }
    if(parametros.size()!=0){
        auxMet.Alerta( comando: "COMANDO REPORT", accion: "No se encontraron algunos parametros, verifique los dat
        return;
```

```cpp
void Reportes::ReporteDisk(string path, string ide) {
    try {
        string direccion;
        Structs::StructParticion particiones = Montar.BusquedaMontar(ide,  path: &direccion);

        FILE *archivo = fopen( filename: direccion.c_str(),  modes: "rb+");
        if (archivo == NULL) {
            throw runtime_error("El disco no se encontro o no existe");
        }

        Structs::StructMBR DiscoAct;
        rewind( stream: archivo);
        fread( ptr: &DiscoAct,  size: sizeof(Structs::StructMBR),  n: 1,  stream: archivo);
        fclose( stream: archivo);

        string aux = path.substr( pos: 0,  n: path.find( c: '.'));
        aux += ".dot";
        FILE *conca = fopen( filename: aux.c_str(),  modes: "r");
        if (conca == NULL) {
            string coman = "mkdir -p \"" + aux + "\"";
            string coman2 = "rmdir \"" + aux + "\"";
            system( command: coman.c_str());
            system( command: coman2.c_str());
        } else {
            fclose( stream: conca);
        }
        Structs::StructParticion particion[4];
        particion[0] = DiscoAct.mbrParticion1;
        particion[1] = DiscoAct.mbrParticion2;
        particion[2] = DiscoAct.mbrParticion3;
        particion[3] = DiscoAct.mbrParticion4;
        Structs::StructParticion extendida;

        bool Auxext = false;
        for (int i = 0; i < 4; ++i) {
            if (particion[i].DisponibilidadParte == '1') {
                if (particion[i].Tipo == 'E') {
                    Auxext = true;
```

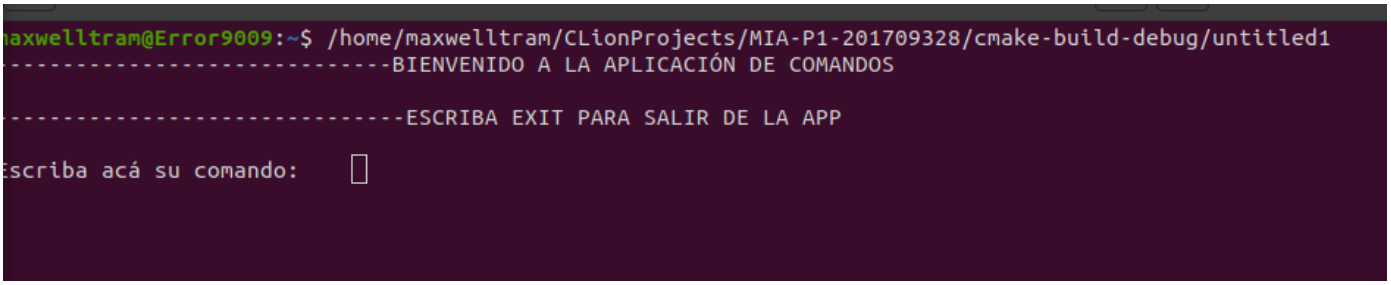


```
void Reportes::ReporteMBR(string path, string ide) {
    try{
        string direccion;
        Structs::StructParticion  particion= Montar.BusquedaMontar(ide,  path: &direccion);
        FILE *archivo = fopen( filename: direccion.c_str(), modes: "rb+");
        if(archivo==NULL){
            throw runtime_error("El disco ingresado, no se ha encontrado");
        }

        Structs::MBRStruct DiscoAct;
        rewind( stream: archivo);
        fread( ptr: &DiscoAct,  size: sizeof(Structs::MBRStruct), n: 1, stream: archivo);
        fclose( stream: archivo);
        string auxp= path.substr( pos: 0, n: path.find( c: '.'));
        auxp+= ".dot";
        FILE *rep= fopen( filename: auxp.c_str(), modes: "r");
        if(rep == NULL){
            string coman= "mkdir -p \""+auxp+"\"";
            string comann= "rmdir \""+auxp+"\"";
            system( command: coman.c_str());
            system( command: comann.c_str());
        }else{
            fclose( stream: rep);
        }

        Structs::StructParticion particiones[4];
        particiones[0] = DiscoAct.mbrParticion1;
        particiones[1] = DiscoAct.mbrParticion2;
        particiones[2] = DiscoAct.mbrParticion3;
        particiones[3] = DiscoAct.mbrParticion4;

        struct tm *tm;

        tm= localtime( timer: &DiscoAct.mbrFechaC);
        char Fechita[20];
        strftime( s: Fechita, maxsize: 20,  format: "%Y/%m/%d %H:%M:%S", tp: tm);
        string contenidoGraph;
```

Clase Reporte, en las cuales hay métodos que genera el reporte a través de la lectura de comandos y luego se crea el reporte a través de la búsqueda de discos, particiones, etc.

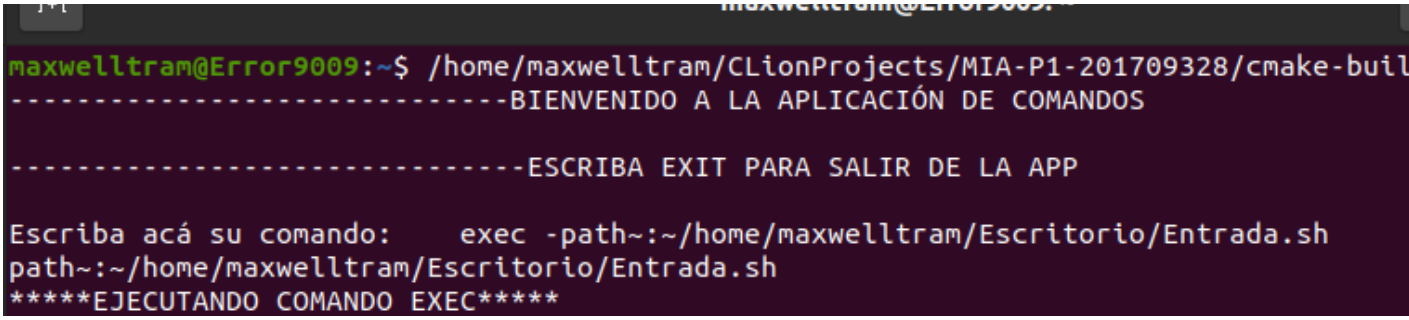
MANUAL DE USUARIO

BIENVENIDA AL PROGRAMA



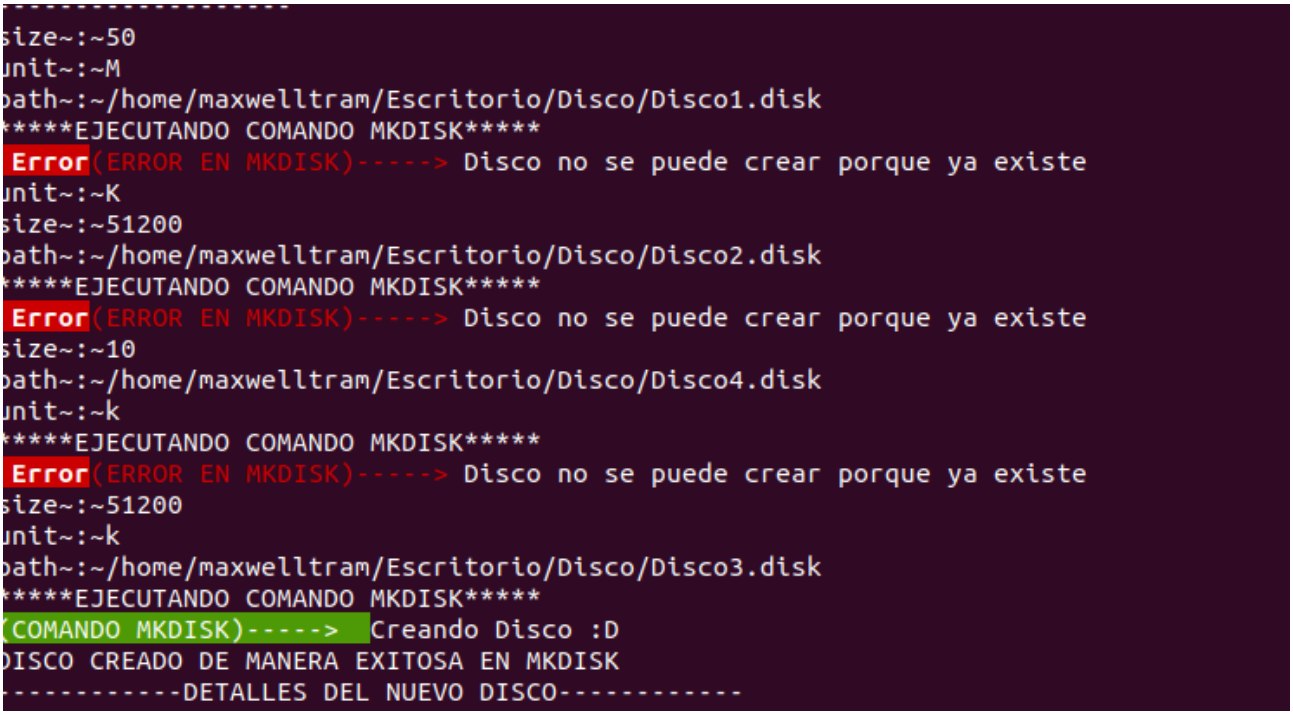
CARGA MASIVA

Se logra a través del comando exec



DISCOS

Se crean discos a través del comando MKDISK, también existe la recuperación de errores como se muestra en la foto.



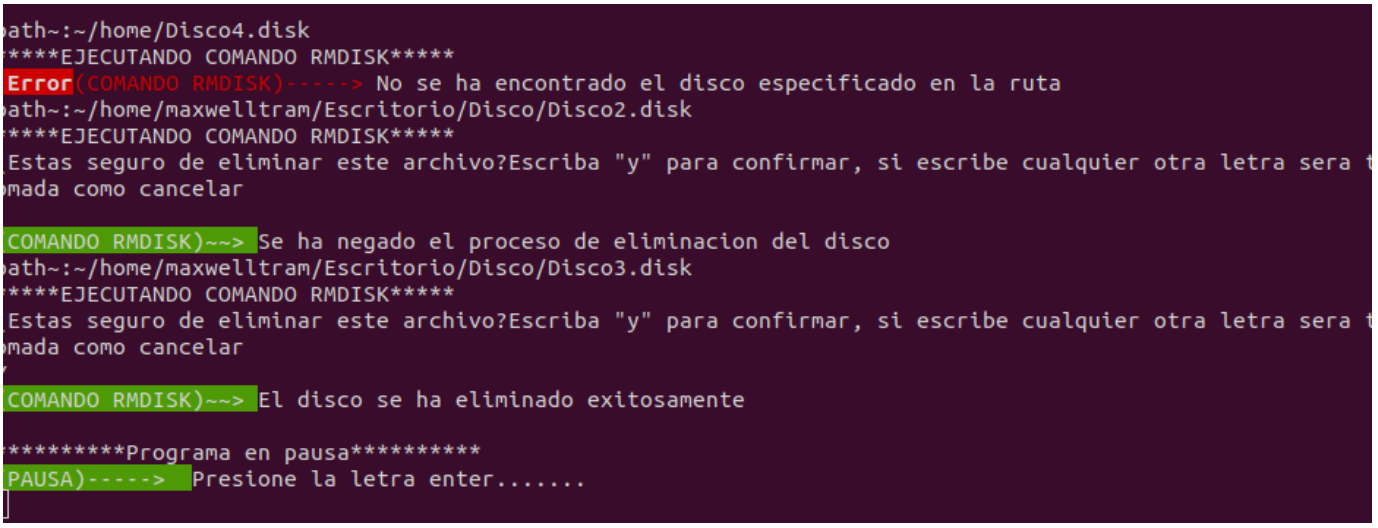
PARTICIONES

Creamos particiones a través del comando FDIKSK

```
 (COMANDO FDISK): La pinche particion fue creada con exito xD
type~:~E
unit~:~K
name~:~Part2
size~:~7680
path~:~/home/maxwelltram/Escritorio/Disco/Disco1.disk
fit~:~WF
*****EJECUTANDO COMANDO FDISK*****
 (COMANDO FDISK): La pinche particion fue creada con exito xD
type~:~E
unit~:~K
name~:~Part3
size~:~7680
path~:~/home/maxwelltram/Escritorio/Disco/Disco1.disk
fit~:~WF
*****EJECUTANDO COMANDO FDISK*****
 Error: (COMANDO FDISK)~> No se puede crear mas particiones logicas, unicamente particiones extendidas
*****EJECUTANDO COMANDO COMENTARIO*****
(COMENTARIO)-----> #Debe dar error por que ya existe una extendida
type~:~P
unit~:~K
name~:~Part3
size~:~7680
path~:~/home/maxwelltram/Escritorio/Disco/Disco1.disk
fit~:~WF
*****EJECUTANDO COMANDO FDISK*****
 (COMANDO FDISK): La pinche particion fue creada con exito xD
type~:~L
unit~:~K
name~:~Part5
size~:~1200
```

RMDISK

Eliminamos discos a través del comando RMDISK



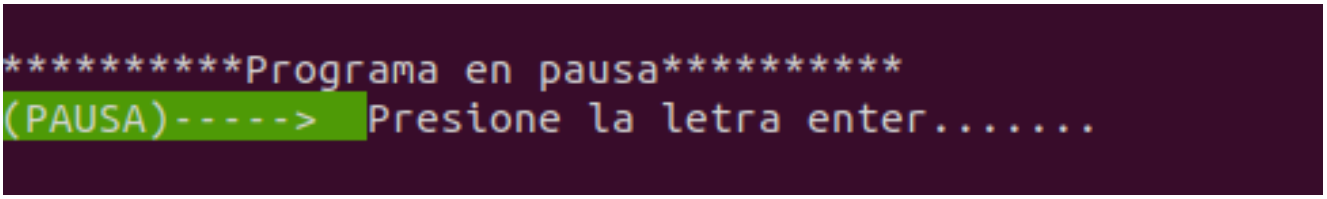
COMENTARIOS

Leemos comentarios a través del uso del signo #

```
*****EJECUTANDO COMANDO EXEC*****
NO SE ENCONTRO NINGUN ERROR EN EL COMANDO SCRIPT
PROBANDO SI FUNCIONA ESTA ONDA
*****EJECUTANDO COMANDO COMENTARIO*****
(COMENTARIO)-----> #Contenido de calificacion.sh
*****EJECUTANDO COMANDO COMENTARIO*****
(COMENTARIO)-----> #Crea 5 discos de 50 Mb
*****EJECUTANDO COMANDO COMENTARIO*****
(COMENTARIO)-----> #/home/maxwelltram/CLionProjects/MIA-P1-201709328/cmake-build-debug/untitled1
*****EJECUTANDO COMANDO COMENTARIO*****
(COMENTARIO)-----> #exec -path~:~/home/maxwelltram/Escritorio/Entrada.sh
*****EJECUTANDO COMANDO COMENTARIO*****
(COMENTARIO)-----> #CREANDO DISCOS ------------------------------------------------------------------------
------------------
```

PAUSA

Dejamos el programa en pausa, a través del uso del comando pause



REPORTE

Creamos reportes de los Discos y particiones a través del comando Report

```
ath~:~/home/maxwelltram/Escritorio/Disco/mbr2.png
ame~:~mbr
(COMANDO REPORTE): Se ha generado el reporte MBR correctamente
d~:~vda1
ath~:~/home/maxwelltram/Escritorio/Disco/disk2.png
ame~:~disk
(COMANDO REPORT): El reporte DISK se genero de manera exitosa
d~:~vdb1
ath~:~/home/maxwelltram/Escritorio/Disco/mbr5.png
ame~:~mbr
Error: (COMANDO REPORTE)~> La partición que se ingreso, no se encuentra
d~:~vdb1
ath~:~/home/maxwelltram/Escritorio/Disco/disk5.png
ame~:~disk
Error: (COMANDO REPORT)~> La partición que se ingreso, no se encuentra
dd~:~-700
nit~:~K
ath~:~/home/maxwelltram/Escritorio/Disco/Disco1.disk
ame~:~Part3
****EJECUTANDO COMANDO FDISK*****
iolación de segmento (`core' generado)
```