

Proyecto Final Linux Terminal

Frida Hernández Rodríguez, Cristopher Velasco Ávila

Abril 2025

Índice

1. Introducción	3
2. Desarrollo	4
2.1. Script Principal: proyecto	4
2.1.1. Captura de pantalla de la ejecución	4
2.1.2. Código fuente del script	4
2.2. Script: ayuda	7
2.2.1. Captura de pantalla de la ejecución	7
2.2.2. Código fuente del script	7
2.3. Script: credits	8
2.3.1. Captura de pantalla de la ejecución	8
2.3.2. Código fuente del script	8
2.4. Script: buscar	9
2.4.1. Captura de pantalla de la ejecución	9
2.4.2. Código fuente del script	9
2.5. Script: infosis	11
2.5.1. Captura de pantalla de la ejecución	11
2.5.2. Código fuente del script	11
2.6. Script: hora	12
2.6.1. Captura de pantalla de la ejecución	12
2.6.2. Código fuente del script	12
2.7. Script: juego	14
2.7.1. Captura de pantalla de la ejecución	14
2.7.2. Código fuente del script	14
2.8. Script: musica	18
2.8.1. Captura de pantalla de la ejecución	18
2.8.2. Código fuente del script	19
2.9. Script: salir	22
2.9.1. Captura de pantalla de la ejecución	22
2.9.2. Código fuente del script	22
3. Conclusiones	23
3.1. Conclusión de Frida	23
3.2. Conclusión de Cristopher	23

1. Introducción

El presente proyecto consiste en el desarrollo de un programa que simula una terminal de trabajo en un sistema operativo basado en Linux. El objetivo principal es crear un entorno personalizado donde el usuario pueda interactuar mediante comandos específicos programados en scripts de Bash. Entre las funciones disponibles se encuentran la visualización de información del sistema, consultar fecha y hora actual, búsqueda de archivos, comando de ayuda para conocer los comandos disponibles, reproducción de música, ejecución de un juego, entre otros.

Entre las particularidades del proyecto se encuentra un sistema de autenticación para acceso a usuarios existentes en el SO anfitrión. El cual hace que al ejecutar el script principal del proyecto, lo primero por hacer es introducir un usuario y contraseña para poder ingresar. De igual manera, se cuenta con una línea de comandos que muestra el usuario activo, el nombre del host y la carpeta donde se encuentra. La terminal es capaz de ejecutar los comandos programados por los miembros de este proyecto así como los disponibles en el SO anfitrión, a excepción del comando `-exit-` el cual fue deshabilitado, al igual que `Ctrl+C` y `Ctrl+Z`, con el objetivo de que solo se pueda salir de la terminal utilizando el comando `-salir-` programado por los integrantes del proyecto.

Cada comando fue desarrollado mediante scripts independientes que son invocados desde un programa principal que gestiona el flujo de la terminal personalizada. Los comandos programados que ejecuta este terminal son los siguientes:

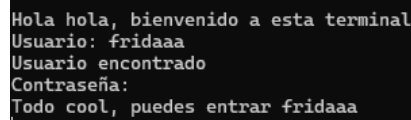
- **proyecto:** Este es el encargado de iniciar el proceso de la terminal y comienza con el login.
- **ayuda:** Comando que muestra una lista de los comandos disponibles y una breve descripción de que es lo que hacen.
- **creditos:** Comando que muestra los nombres de los integrantes del proyecto.
- **buscar:** Este comando busca un archivo en una carpeta.
- **infosis:** Comando que muestra la información del sistema:
- **hora:** Comando que muestra la fecha y hora actual.
- **juego:** Comando que inicia el juego gato.
- **musica:** Comando que arroja el reproductor mp3.
- **salir:** Comando que termina el proceso de `-proyecto-` es decir, la terminal personalizada.

2. Desarrollo

2.1. Script Principal: proyecto

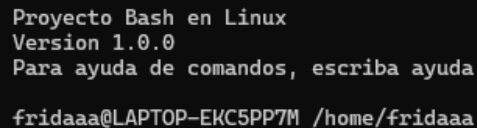
El script llamado -proyecto- constituye el script principal del proyecto, actuando como una terminal personalizada. Permite al usuario autenticarse en el sistema, proporciona un menú de comandos personalizados y controla la interacción con los demás scripts. Este script se ejecuta cuando el usuario ingresa `./proyecto.sh` en la terminal.

2.1.1. Captura de pantalla de la ejecución



```
Hola hola, bienvenido a esta terminal
Usuario: fridaaa
Usuario encontrado
Contraseña:
Todo cool, puedes entrar fridaaa
```

Figura 1: Ejecución del script proyecto - LOGIN



```
Proyecto Bash en Linux
Version 1.0.0
Para ayuda de comandos, escriba ayuda

fridaaa@LAPTOP-EKC5PP7M /home/fridaaa -> |
```

Figura 2: Ejecución del script de proyecto - Linea de comandos

2.1.2. Código fuente del script

```
# !/bin/bash
# Proyecto de bash

trap '' SIGINT
trap '' SIGTSTP

# LOGIN
tput clear
echo "Hola hola , bienvenido a esta terminal"
#Usuario
```

```

read -p "Usuario:-" usuario

if id "$usuario" &>/dev/null; then
    echo "Usuario encontrado"
else
    echo "No se encontro el usuario"
    exit 1
fi

#Contrasena
read -s -p Contraseña:
echo

if [ -z "$contrasena" ]; then
    echo "No ingresaste ninguna contraseña.- Intenta de nuevo."
    exit 1
fi

echo "$contrasna" | sudo -S -u "$usuario" whoami &>/dev/null

if [ $? -eq 0 ] ; then
    echo "Todo cool , puedes entrar - $usuario"
else
    echo "- Ey te equivocaste en la contraseña , no puedes entrar , intenta de nuevo"
    exit 1
fi

sleep 2
clear

command=" null"
guia=$(pwd)
cd ~

export PROYECTO_PID=$$

echo " Proyecto - Bash - en - Linux"
echo " Version - 1.0.0"
echo " Para ayuda de comandos , escriba - "ayuda""
echo
sleep 2

while true;
do
    read -p "${usuario}@${HOSTNAME} - $(pwd) ->- " command

```

```

case $command in

ayuda)
    echo
    $guia/comandos/ayudaa.sh
    echo
    ;;

creditos)
    $guia/comandos/creditos.sh
    ;;

buscar)
    $guia/encuentra.sh
    ;;

infosis)
    $guia/infosis.sh
    ;;

hora)
    $guia/hora.sh
    ;;

juego)
    $guia/gato.sh
    ;;

musica)
    echo "Aqui va el reproductor.sh"
    ;;

salir)
    $guia/comandos/salir.sh
    ;;

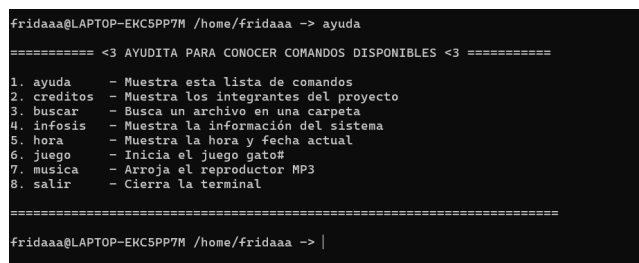
exit)
    echo "Comando deshabilitado.- Usa el comando --salir -- para cerrar
    ----- la terminal"
    ;;
*)
    $command
    ;;
esac
done

```

2.2. Script: ayuda

El script de ayuda proporciona al usuario una lista de los comandos disponibles dentro de la terminal simulada, junto con una breve descripción de cada uno. Este script se ejecuta cuando el usuario ingresa el comando `ayuda` en la terminal.

2.2.1. Captura de pantalla de la ejecución



```
fridaaa@LAPTOP-EKC5PP7M /home/fridaaa -> ayuda
===== <3 AYUDITA PARA CONOCER COMANDOS DISPONIBLES <3 =====
1. ayuda      - Muestra esta lista de comandos
2. creditos   - Muestra los integrantes del proyecto
3. buscar     - Busca un archivo en una carpeta
4. infosis    - Muestra la información del sistema
5. hora       - Muestra la hora y fecha actual
6. juego      - Inicia el juego gato#
7. musica     - Arroja el reproductor MP3
8. salir      - Cierra la terminal
=====
fridaaa@LAPTOP-EKC5PP7M /home/fridaaa -> |
```

Figura 3: Ejecución del script de ayuda

2.2.2. Código fuente del script

```
#!/bin/bash
```

```
# Script de ayuda para mostrar comandos disponibles
```

```
echo "===== <3 AYUDITA PARA CONOCER COMANDOS DISPONIBLES <3 ====="
echo ""
echo "1. ayuda      - Muestra esta lista de comandos"
echo "2. creditos   - Muestra los integrantes del proyecto"
echo "3. buscar     - Busca un archivo en una carpeta"
echo "4. infosis    - Muestra la información del sistema"
echo "5. hora       - Muestra la hora y fecha actual"
echo "6. juego      - Inicia el juego gato#"
echo "7. musica     - Arroja el reproductor MP3"
echo "8. salir      - Cierra la terminal"
echo ""
echo "=====
```

2.3. Script: credits

El script de créditos tiene como propósito mostrar de manera visual y estilizada los créditos del proyecto, destacando a los desarrolladores de forma artística mediante figuras de texto ASCII. El script no recibe parámetros de entrada ni requiere interacción del usuario. Su única función es desplegar la información visualmente atractiva en la terminal. Este script se ejecuta cuando el usuario ingresa el comando `creditos` en la terminal.

2.3.1. Captura de pantalla de la ejecución



Figura 4: Ejecución del script de creditos

2.3.2. Código fuente del script

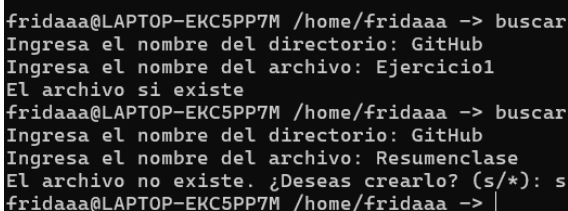
```
#!/bin/bash
```

[illegible]

2.4. Script: buscar

El script de buscar permite al usuario verificar si un archivo existe dentro de un directorio específico. Si el archivo no existe, ofrece la opción de crearlo automáticamente en la ruta indicada. Su propósito es facilitar la gestión básica de archivos mediante una interfaz sencilla de línea de comandos, mejorando la eficiencia y la interacción del usuario con el sistema de archivos. Este script se ejecuta cuando el usuario ingresa el comando **buscar** en la terminal.

2.4.1. Captura de pantalla de la ejecución



```
fridaaaa@LAPTOP-EKC5PP7M /home/fridaaaa -> buscar
Ingresa el nombre del directorio: GitHub
Ingresa el nombre del archivo: Ejercicio1
El archivo si existe
fridaaaa@LAPTOP-EKC5PP7M /home/fridaaaa -> buscar
Ingresa el nombre del directorio: GitHub
Ingresa el nombre del archivo: Resumenclase
El archivo no existe. ¿Deseas crearlo? (s/*): s
fridaaaa@LAPTOP-EKC5PP7M /home/fridaaaa -> |
```

Figura 5: Ejecución del script de buscar

2.4.2. Código fuente del script

```
#!/bin/bash

#Leemos el directorio
read -p "Ingresa el nombre del directorio: -" path

#Ingresa diagonal al final de path si no la tiene
if [[ ! ${path} == */ ]]; then
    path="${path}/"
fi

#Revisamos si el directorio existe
if [[ ! -d ${path} ]]; then
    echo "-Error! -El directorio no existe.-Abortando..."
    exit
fi

#Leemos el nombre del archivo
read -p "Ingresa el nombre del archivo: -" name
```

```

#Revisamos si el archivo existe como tal
if [ -e "${path}${name}" ]; then
    echo "El archivo si existe"
else
    read -p "El archivo no existe.-Deseas crearlo?(s/*):-" opcion
    if [[ $opcion == "s" || $opcion == "S" ]]; then
        touch ${path}${name}
    fi
fi

```

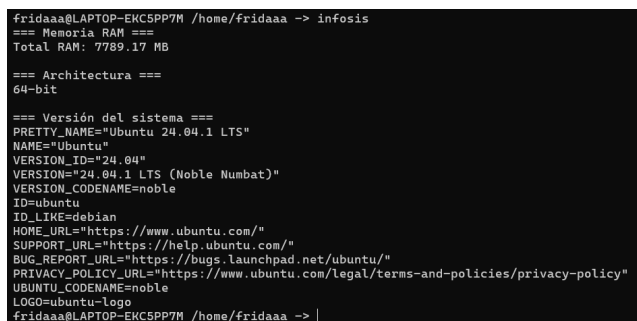
2.5. Script: infosis

El script de infosis muestra de manera sencilla y rápida tres tipos de información sobre el sistema operativo:

- Cantidad de Memoria RAM disponible
- Arquitectura del procesador (si es de 32 o 64 bits)
- Versión y detalles del sistema operativo

Es muy útil para hacer diagnósticos básicos o simplemente para conocer las características de la computadora donde se está trabajando. Este script se ejecuta cuando el usuario ingresa el comando `infosis` en la terminal.

2.5.1. Captura de pantalla de la ejecución



```
fridaaa@LAPTOP-EKCSPP7M /home/fridaaa -> infosis
=== Memoria RAM ===
Total RAM: 7789.17 MB

=== Arquitectura ===
64-bit

=== Versión del sistema ===
PRETTY_NAME="Ubuntu 24.04.1 LTS"
NAME="Ubuntu"
VERSION_ID="24.04"
VERSION="24.04.1 LTS (Noble Numbat)"
VERSION_CODENAME=noble
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
UBUNTU_CODENAME=noble
LOGO=ubuntu-logo
fridaaa@LAPTOP-EKCSPP7M /home/fridaaa -> |
```

Figura 6: Ejecución del script de infosis

2.5.2. Código fuente del script

```
#!/bin/bash

echo "===- Memoria -RAM-==="
awk '/MemTotal/{ print "- Total -RAM: -" - $2/1024 -" -MB"}' /proc/meminfo

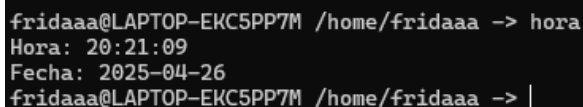
echo -e "\n===- Arquitectura -==="
cat /proc/cpuinfo | grep flags | grep -q '-lm-' && echo "64-bit" ||
echo "32-bit"

echo -e "\n===- Version -del -sistema -==="
cat /etc/os-release
```

2.6. Script: hora

El script de hora lee la hora y la fecha actuales desde el archivo `/proc/driver/rtc`, que contiene información sobre la hora del sistema. Luego, ajusta la hora en función de una zona horaria específica, en este caso, la zona horaria con un offset de -6 horas. El script maneja la conversión de hora y fecha, ajustando tanto la hora como el día si es necesario (por ejemplo, si la hora ajustada es menor a 0 o mayor a 23). Finalmente, muestra la hora y la fecha ajustadas en formato HH:MM:SS y AAAA:MM:DD. El objetivo de este script es poder ajustar la hora y fecha del sistema a una zona horaria diferente a la local. Este script se ejecuta cuando el usuario ingresa el comando `hora` en la terminal.

2.6.1. Captura de pantalla de la ejecución



```
fridaaa@LAPTOP-EKC5PP7M /home/fridaaa -> hora
Hora: 20:21:09
Fecha: 2025-04-26
fridaaa@LAPTOP-EKC5PP7M /home/fridaaa -> |
```

Figura 7: Ejecución del script de hora

2.6.2. Código fuente del script

```
#!/bin/bash

# Leemos hora y fecha desde el archivo /proc/driver/rtc
hora=$(grep -o -E -m 1 '[0-9]{2}:[0-9]{2}:[0-9]{2}' /proc/driver/rtc)
fecha=$(grep -o -E -m 1 '[0-9]{4}-[0-9]{2}-[0-9]{2}' /proc/driver/rtc)

# Separamos la hora en sus tres componentes
horas=${hora:0:2}
minutos=${hora:3:2}
segundos=${hora:6:2}

# Separamos la fecha en tres componentes
anio=${fecha:0:4}
meses=${fecha:5:2}
dias=${fecha:8:2}

# Zona horaria nueva
```

```

offset=-6

hora_nueva=$((10#$horas + offset))

# Ajustar fecha si es necesario

if [ "$hora_nueva" -lt 0 ]; then
    hora_nueva=$(( hora_nueva + 24))
    dias=$((10#$dias - 1))
fi

if [ "$hora_nueva" -ge 24 ]; then
    hora_nueva=$(( hora_nueva - 24))
    dias=$((10#$dias + 1))
fi

echo "Hora: ${hora_nueva}:${minutos}:${segundos}"
echo "Fecha: ${anio}-${meses}-${dias}"

```

2.7. Script: juego

El script de juego implementa el clásico juego de Gato para dos jugadores en la terminal. Usa un arreglo para representar el tablero y alterna turnos entre las marcas X, O. Cada jugador elige una posición disponible, validándose que sea un movimiento válido. Tras cada turno, se verifica si hay un ganador o si se produce un empate. El tablero se actualiza y muestra en cada jugada, permitiendo seguir el progreso del juego. Este script se ejecuta cuando el usuario ingresa el comando `juego` en la terminal.

2.7.1. Captura de pantalla de la ejecución

```
fridaaa@LAPTOP-EKC5PP7M /home/fridaaa -> juego
Bienvenido a Gato
-----
Turno numero 1! toca a X
Tablero actual:
# | # | #
-----
# | # | #
-----
# | # | #
Elige posicion (1-9): 5
-----
Turno numero 2! toca a O
Tablero actual:
# | # | #
-----
# | X | #
-----
# | # | #
Elige posicion (1-9): 2
-----
Turno numero 3! toca a X
Tablero actual:
# | O | #
-----
# | X | #
-----
# | # | #
Elige posicion (1-9): 4
-----
Turno numero 4! toca a O
Tablero actual:
# | O | #
-----
X | X | #
-----
# | # | #
Elige posicion (1-9): |
```

Figura 8: Ejecución del script de juego

2.7.2. Código fuente del script

```
#!/bin/bash
```

```

echo "Bienvenido a Gato"

tablero=("#" "#" "#" "#" "#" "#" "#" "#")

function imprimir_tablero() {
    echo "Tablero actual:"
    echo "${tablero[0]} - | - ${tablero[1]} - | - ${tablero[2]}"
    echo "_____"
    echo "${tablero[3]} - | - ${tablero[4]} - | - ${tablero[5]}"
    echo "_____"
    echo "${tablero[6]} - | - ${tablero[7]} - | - ${tablero[8]}"
}

function revisar_ganador(){

    # Check de horizontales
    if [[ ${tablero[0]} == ${tablero[1]} &&
        ${tablero[1]} == ${tablero[2]} && ${tablero[0]} != "#" ]]; then
        ganador="${tablero[0]}"
    else if [[ ${tablero[3]} == ${tablero[4]} && ${tablero[4]} == ${tablero[5]}
        && ${tablero[3]} != "#" ]]; then
        ganador="${tablero[3]}"
    else if [[ ${tablero[6]} == ${tablero[7]} && ${tablero[7]} == ${tablero[8]}
        && ${tablero[6]} != "#" ]]; then
        ganador="${tablero[6]}"
    # Check de verticales
    else if [[ ${tablero[0]} == ${tablero[3]} && ${tablero[3]} == ${tablero[6]}
        && ${tablero[0]} != "#" ]]; then
        ganador="${tablero[0]}"
    else if [[ ${tablero[1]} == ${tablero[4]} && ${tablero[4]} == ${tablero[7]}
        && ${tablero[1]} != "#" ]]; then
        ganador="${tablero[1]}"
    else if [[ ${tablero[2]} == ${tablero[5]} && ${tablero[5]} == ${tablero[8]}
        && ${tablero[2]} != "#" ]]; then
        ganador="${tablero[2]}"
    # Check de diagonales
    else if [[ ${tablero[0]} == ${tablero[4]} && ${tablero[4]} == ${tablero[8]}
        && ${tablero[0]} != "#" ]]; then
        ganador="${tablero[0]}"
    else if [[ ${tablero[2]} == ${tablero[4]} && ${tablero[4]} == ${tablero[6]}
        && ${tablero[2]} != "#" ]]; then
        ganador="${tablero[2]}"
    else if [[ ${numeroTurno} == 10 ]]; then
        ganador="Empate"
    fi fi fi fi fi fi fi fi fi

```

```

}

ganador=0
turnoActual=X
numeroTurno=1

while [ $ganador == 0 ]
do

    if [[ ${turnoActual} == "X" && ${error} == 0 ]]; then
        turnoActual="O"
    else if [[ ${turnoActual} == "O" && ${error} == 0 ]]; then
        turnoActual="X"
    fi fi

    echo "_____ "
    echo "Turno-numero-$numeroTurno!-toca-a-${turnoActual}"
    imprimir_tablero

    error=0

    read -p "Elige-posicion-(1-9):-" posicion

    if [[ posicion -gt 9 || posicion -lt 1 ]]; then
        echo "Posicion-no-valida,-elige-otra!"
        error=1
    else if [ ! ${tablero[$posicion-1]} == "#" ]; then
        echo "Posicion-ocupada,-elige-otra!"
        error=1
    else if [ ${tablero[$posicion-1]} == "#" ]; then
        tablero[$posicion-1]="${turnoActual}"
    fi fi fi

    if [ $error == 0 ]; then
        numeroTurno=$(( numeroTurno + 1 ))
    fi

    revisar_ganador

done

echo "_____ "
imprimir_tablero

```



```
if [[ ${ganador} == "Empate" ]]; then  
    echo "Empate! - Nadie gana"  
    echo  
else  
    echo "El ganador es - $ganador!"  
    echo  
fi  
exit
```

2.8. Script: musica

El script de musica actúa como un reproductor de música en la terminal, permitiendo al usuario seleccionar y reproducir canciones MP3 desde un directorio específico. Pide al usuario el nombre del directorio, verifica su existencia, y muestra una lista de canciones. Luego, permite reproducir canciones con comandos para avanzar, retroceder o salir, y continúa hasta que el usuario decida salir. Este script se ejecuta cuando el usuario ingresa el comando `musica` en la terminal.

2.8.1. Captura de pantalla de la ejecución

```
Proyecto Bash en Linux
Version 1.0.0
Para ayuda de comandos, escriba ayuda

crizz@uka-uka /home/crizz -> musica
Ingresa el nombre del directorio de musica (debe estar ubicado en /home/crizz!): Música
```

Figura 9: Ejecución del script de musica

```
=== REPRODUCTOR MP3 ===
CANCIONES ENCONTRADAS :D
[0] Alex Moukala - Amalgamaniac.mp3
[1] Alex Moukala - Eldritch House.mp3
[2] Alex Moukala - Om And On.mp3
[3] Alex Moukala - RAVEVENGE.mp3

Ingresa el indice de la cancion: 0
Controles: [K] - Anterior, [L] - Siguiente, [Q] - Salir

Playing: Alex Moukala - Amalgamaniac.mp3

Playing: Alex Moukala - Eldritch House.mp3

Playing: Alex Moukala - Om And On.mp3

Playing: Alex Moukala - RAVEVENGE.mp3
Playlist finalizada. Hasta luego!
crizz@uka-uka /home/crizz ->
```

Figura 10: Ejecución del script de musica

```

=== REPRODUCTOR MP3 ===
CANCIONES ENCONTRADAS :D
[0] Alex Moukala - Amalgamaniac.mp3
[1] Alex Moukala - Eldritch House.mp3
[2] Alex Moukala - On And On.mp3
[3] Alex Moukala - RAVEVENGE.mp3

Ingresa el indice de la cancion: 0
Controles: [K] - Anterior, [L] - Siguiente, [Q] - Salir

Playing: Alex Moukala - Amalgamaniac.mp3

```

Figura 11: Ejecución del script de musica

2.8.2. Código fuente del script

```

#!/bin/bash
#Pregunta nombre de carpeta
read -p "Ingresa el nombre del directorio de musica (debe estar ubicado en
/home/${USERNAME}):" carpeta

#Guarda la carpeta en MUSIC_DIR
MUSIC_DIR="/home/${USERNAME}/${carpeta}"
clear

if [ ! -d "$MUSIC_DIR" ]; then
    echo "Error! no existe tal directorio!"
    exit 1
fi

# Guardamos MP3 a un arreglo
CANCIONES=("$MUSIC_DIR"/*.mp3)
NO_CANCIONES=${#CANCIONES[@]}

# Mostramos la lista de canciones encontradas
echo "=== REPRODUCTOR MP3 ==="
echo "CANCIONES ENCONTRADAS :D"
for i in "${!CANCIONES[@]}"; do #Buscamos por indices, no por el nombre
    echo "- [ $i ] - $(basename "${CANCIONES[ $i ]}")"
done

echo
read -p "Ingresa el indice de la cancion:" INDICE_INI

#Verificamos si el indice esta dentro del intervalo valido
if [[ "$INDICE_INI" -lt 0 || "$INDICE_INI" -ge "$NO_CANCIONES" ]]; then

```

```

        echo "Invalid indice."
        exit 1
    fi

#Imprimimos controles
    echo "Controles: [K] -- Anterior, [L] -- Siguiente, [Q] -- Salir"

    indice=$INDICE_INI

    while [ $indice -ge 0 ] && [ $indice -lt $NO_CANCIONES ]; do
        CANCION_ACTUAL=$(basename "${CANCIONES[$indice]}")
        echo -e "\nPlaying: $CANCION_ACTUAL"

        #Reproducir cancion
        mpg123 -q "${CANCIONES[$indice]}" &
        PID=$!
        actual=""

        #Verificamos si el usuario escribe algo en teclado
        while kill -0 $PID 2>/dev/null; do
            read -rsn1 -t 0.1 key
            if [[ $key == "l" ]]; then
                actual="siguiente"
                break
            elif [[ $key == "k" ]]; then
                actual="anterior"
                break
            elif [[ $key == "q" ]]; then
                kill $PID
                wait $PID 2>/dev/null
                echo "Saliendo.-Adios!"
                exit 0
            fi
        done

        #Se realiza accion segun la entrada de usuario
        if [[ $actual == "siguiente" ]]; then
            kill $PID
            wait $PID 2>/dev/null
            indice=$((indice + 1))

            elif [[ $actual == "anterior" ]]; then
                kill $PID
                wait $PID 2>/dev/null
                if [ $indice -gt 0 ]; then

```

```

        indice=$(( indice - 1 ))
    else
        echo "Te encuentras en la primer cancion!"
        indice=0
    fi
else
    wait $PID
    indice=$(( indice + 1 ))
fi
done

echo "Playlist finalizada .- Hasta luego!"

```

2.9. Script: salir

El script de salir está diseñado para mostrar un mensaje visual en el terminal y luego finalizar un proceso asociado a una variable de entorno. En la primera parte, el script imprime una serie de cadenas de texto que forman una representación artística (un banner) utilizando caracteres ASCII. Tras mostrar el banner, el script espera unos segundos, y luego verifica si existe un valor en la variable de entorno. Si existe un proceso con ese ID, el script lo termina usando el comando kill-9. Este script se ejecuta cuando el usuario ingresa el comando `salir` en la terminal.

2.9.1. Captura de pantalla de la ejecución



Figura 12: Ejecución del script de salir

2.9.2. Código fuente del script

```
#!/bin/bash

echo ""
echo "CERRANDO TERMINAL, -BYE"
echo ""
sleep 2

if [[ -n "PROYECTO_PID" ]]; then
    kill -9 $PROYECTO_PID
else
    echo "ERROR"
fi
```

3. Conclusiones

3.1. Conclusión de Frida

Este proyecto ha sido una experiencia significativa en muchos aspectos. En primer lugar, aprendí a trabajar con Bash y a entender mejor el funcionamiento interno de una terminal de Linux. Sin embargo, también me permitió desarrollar mis habilidades de trabajo en equipo, ya que tanto mi compañero como yo tuvimos que aplicar conceptos clave como organización, priorización, comunicación y responsabilidad. Estos elementos fueron esenciales para poder coordinar las tareas de manera efectiva, asegurándonos de que todas las partes del proyecto se completaran de acuerdo a lo planificado y dentro de los plazos establecidos.

Aunque nunca antes había tenido contacto con Linux, este proyecto me dio la oportunidad de aprender y profundizar en el uso de un sistema operativo que antes me resultaba desconocido. Desde la instalación y configuración del entorno hasta la implementación de comandos personalizados, cada paso me permitió mejorar mis habilidades técnicas. Me enfrenté a varios retos, como implementar el sistema de inicio de sesión y estructurar adecuadamente los scripts, pero después de varios intentos logré resolverlos. Considero que este proyecto fortaleció mis habilidades en programación de scripts y en documentación técnica mediante LaTeX. Pero más allá de los aspectos técnicos, lo que más me impactó fue el desarrollo de mis habilidades prácticas. La constante interacción con el sistema me permitió practicar y entender conceptos clave lo cual me ha proporcionado una base sólida para proyectos futuros.

En resumen, este proyecto no solo me ha permitido adquirir nuevos conocimientos en Linux y programación, sino que también me ha ayudado a fortalecer mi capacidad de trabajar en equipo, coordinar tareas, y afrontar desafíos con responsabilidad. Estoy segura de que esta experiencia será valiosa para mi desarrollo profesional y académico, y me ha motivado a seguir aprendiendo y enfrentando nuevos retos.

3.2. Conclusión de Cristopher

El desarrollo de habilidades con sistemas operativos basados en Unix es muy importante a la hora de trabajar en cualquier ámbito. Funciona muy bien como un sistema como alternativa al típico Windows y algunas distribuciones pueden ofrecer muchas mejoras frente al sistema que siempre se utiliza. No solo es importante tener instalado el sistema en si mismo. Si bien el manejo es muy similar al de cualquier otro sistema Windows, si se aprende a usar herramientas mas especificas dentro del sistema como el manejo de la terminal, de los servicios, y de los scripts en bash, pueden resultar una herramienta muchísimo mas practica y eficaz al momento de trabajar. Es bien sabido que en Windows, a la hora de tener que instalar nuevas herramientas como compiladores o programas, se deben de seguir cierta serie de pasos específicos que en ciertas ocasiones pueden resultar confusas de seguir, y otra característica importante de Linux es que este tipo de problemas se eliminan por completo a la hora de utilizar un gestor de

paquetes. Desde luego que usar un sistema Linux, y comprender la mayoría de las funciones básicas son habilidades que cualquier usuario debería de aprender para que su productividad mejore significativamente.

Finalmente, cabe destacar que hay ciertas situaciones donde usar un sistema diferente como Windows podría resultar en un resultado mejor, algo muy típico y que siempre se discute es a la hora de usar la suite de aplicaciones de Adobe, sin embargo, se trata de casos particulares. La cantidad de información que se puede encontrar en internet acerca del uso de este tipo de sistemas es bastante extensa, con una gran cantidad de foros, artículos y enciclopedias que cualquier usuario, tanto novato como experimentado puede consultar, por lo tanto, los conocimientos adquiridos servirán mucho en el futuro para diversas actividades, desde el desarrollo de aplicaciones y programación, hasta cosas mas sencillas como la edición de documentos, uso de aplicaciones de ofimática o navegar en internet.