



PROTECSA

Proyecto final GNU/LINUX

Montiel Juarez, Guillermo Jair¹ and Rincón Villa, Gretchen Itzel²

¹Facultad de Contaduría, UNAM

²Facultad de Ciencias, UNAM

26 Abril 2025

Resumen :

El proyecto final de LINUX en el curso de PROTECSA recopila todo el aprendizaje para ejecutar un entorno de terminal dentro de la misma terminal del sistema, a la que llamaremos la “terminal emulada” o “emulador”. Para su realización, se programó enteramente en bash, y se encuentra organizado de forma modular, cada comando es un nuevo script que es llamado desde "terminal.sh" lo cual es útil para agregar y modificar comandos sin corromper el programa original. Este programa está repartido a través de diferentes módulos, y realizado enteramente desde nuestras terminales, cada cambio está registrado como un commit en Github.

Índice

1. Script terminal.sh	1
1.1. Script infosis.sh	4
1.2. Script fecha.sh	6
1.3. Script busca.sh	6
1.4. Script creditos.sh	7
1.5. Script jugar.sh	8
1.6. Script musica.sh	11
2. Conclusiones por Buddy	15
2.1. Jair Montiel	15
2.2. Gretchen Rincón	15

Índice de figuras

1. Ingreso y salida exitosas de la terminal emulada	2
2. Uso del comando ayuda	4
3. Uso del comando infosis	6
4. Uso del comando fecha	6
5. Enter Caption	7
6. Uso comando creditos	8
7. Uso del comando jugar, se repitió hasta ganar en 3 movimientos para esta captura .	11
8. Interfaz gráfica del reproductor de música	13
9. Menu interno de mpg123 usando la interfaz whiptail	13
10. Menu para escuchar otra canción	14

1. Script terminal.sh

Este script implementa un emulador de terminal de Linux escrito en Bash, que al iniciar solicita credenciales de inicio de sesión para ingresar. Las credenciales, que son el usuario y la contraseña, deben de coincidir con las que existan en el sistema, pues de otra forma, no ejecutará la terminal. Una vez se ingresa, se abre el ejecutable, que estará en constante ejecución, hasta que se decida salir de la terminal con el comando “salir”, y sólo con este comando, pues se bloqueó la entrada del `ctrl+z` y `ctrl+x` para que el usuario no pueda terminar la ejecución del script. El emulador es capaz de ejecutar los comandos que tiene el sistema por default, utilizando el comando `eval` junto con un string ingresado por el mismo usuario, y ejecuta otros comandos personalizados, que deben de coincidir con algún archivo bash dentro de la misma carpeta del proyecto. Estos comandos no están predefinidos, por lo que el programador puede ir agregando más comandos adicionales si lo desea, siempre y cuando coincida el nombre del archivo con el del archivo bash (sin la terminación “.sh”).

```

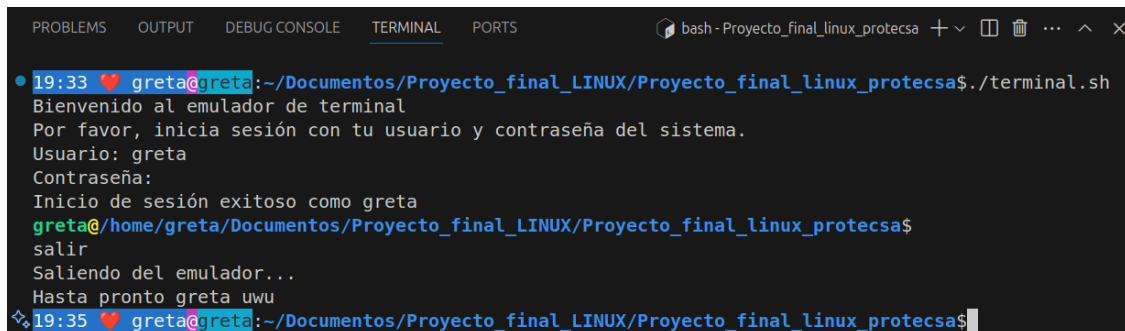
1  #!/bin/bash
2
3  ruta=$(pwd) # Ruta del script actual
4
5  #Emulador de una terminal de Linux
6  echo "Bienvenido al emulador de terminal"
7
8  # Captura Ctrl+C y Ctrl+Z para evitar salida
9  trap '' SIGINT SIGTSTP
10
11 #Función para iniciar sesión con el usuario del sistema
12 login(){
13     # Solicitar usuario y contraseña
14     echo "Por favor, inicia sesión con tu usuario y contraseña del sistema."
15     read -p "Usuario: " usuario
16     read -s -p "Contraseña: " password
17     echo ""
18     # Verificar si el usuario y contraseña existen en el sistema con id
19     # id devuelve true si el usuario existe
20     if id "$usuario" &>/dev/null; then
21         # Verificar la contraseña
22         if echo "$password" | su -c "true" "$usuario" &>/dev/null; then
23             echo "Inicio de sesión exitoso como $usuario"
24         else
25             echo "Contraseña incorrecta"
26             exit 1
27         fi
28     else
29         echo "Usuario no encontrado"
30         exit 1
31     fi
32 }
33
34 # Función para mostrar la línea de comandos personalizada
35 mostrar_prompt() {
36     directorio_actual=$(pwd)
37     #echo -e "\033[1;32m$usuario\033[0m@\033[1;34m$directorio_actual\033[0m$ "
38     echo -e
39     ↵ "\033[1;32m$usuario\033[0m@\033[1;33m@\033[0m\033[1;34m$directorio_actual\033[0m$
40     ↵ "
41 }

```

```

41
42 # Función para ejecutar comandos en el emulador de terminal
43 ejecutar_terminal(){
44     #La función se ejecuta en un bucle infinito hasta que el usuario decida salir
45     while true; do
46         mostrar_prompt
47         # Leer el comando ingresado por el usuario
48         # Usar 'read -e' para permitir la edición del comando
49         read -e comando
50
51         # Bloquear la palabra "exit"
52         if [[ "$comando" == "exit" ]]; then
53             echo "Usa 'salir' para salir del emulador."
54             continue
55         fi
56
57         # Comprobar si el comando ingresado corresponde a un archivo .sh en la ruta
58         if [ -f "$ruta/$comando.sh" ]; then
59             # El comando se ejecuta en un subshell para evitar conflictos con el entorno
60             ↪ actual
61             bash "$ruta/$comando.sh"
62             continue
63         fi
64
65         # Salir del emulador si el usuario escribe "salir"
66         if [[ "$comando" == "salir" ]]; then
67             echo "Saliendo del emulador..."
68             echo "Hasta pronto $usuario uwu"
69             break
70         fi
71
72         # Ejecutar el comando ingresado por el usuario
73         # eval es utilizado para evaluar una cadena de texto y ejecutarla
74         eval "$comando"
75     done
76 }
77
78 #.....
79
80 # Ejecutar la función principal
81 login
82 ejecutar_terminal

```



```

19:33 greta@greta:~/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$ ./terminal.sh
Bienvenido al emulador de terminal
Por favor, inicia sesión con tu usuario y contraseña del sistema.
Usuario: greta
Contraseña:
Inicio de sesión exitoso como greta
greta@/home/greta/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$
salir
Saliendo del emulador...
Hasta pronto greta uwu
19:35 greta@greta:~/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$

```

Figura 1: Ingreso y salida exitosas de la terminal emulada

Script ayuda.sh

Este script despliega un menú de 8 comandos especiales. Su objetivo es mostrar la información de lo que realiza cada comando para que sepas que esperar de la forma más concreta posible. Comienza con un mensaje de bienvenida, luego entra en un bucle infinito que solo se rompe al escribir el comando "salir". El menú permite al usuario ingresar cualquier cadena de texto pero solo arroja información sobre "ayuda", "infosis", "fecha", "busca", "creditos", "jugar" y "musica".

```

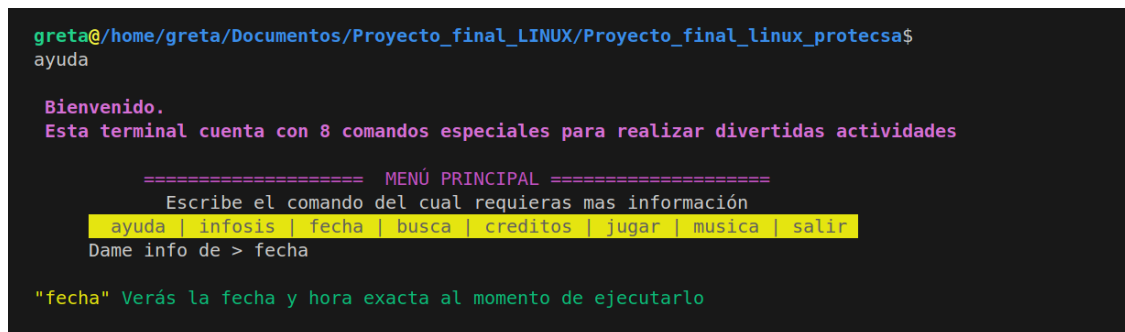
1  #!/bin/bash
2
3
4  #trap es un comando de bash para capturar señales y asignar una acción personalizada
5  #' indica una acción vacía, SIGINT es la señal Ctrl+C, SIGTSTP es la señal Ctrl+Z
6  trap '' SIGINT SIGTSTP
7
8  echo -e "\033[5;1;35m \n Bienvenido. \n Esta terminal cuenta con 8 comandos especiales
↳ para realizar divertidas actividades \033[0m"
9
10 #Todo el script funcionara con un bucle que solo el comando "salir" detiene
11 #Desplegamos un menú de comandos especiales disponibles
12 #La bandera -e en un echo sirve para interpretar saltos de línea \n, barras invertidas
↳ \|, etc. Previene errores de sintaxis
13
14 while true; do
15     echo -e ""
16     echo -e "\033[35m          ===== MENÚ PRINCIPAL
↳ =====\033[0m"
17     echo -e "          Escribe el comando del cual requieras mas información"
18     echo -e "          \033[43m ayuda | infosis | fecha | busca | credits | jugar | musica
↳ | salir \033[0m"
19     read -p "          Dame info de > " comando
20     echo -e ""
21
22     case $comando in
23
24         "ayuda")
25             echo -e "\033[33m\"ayuda\" \033[32m es el primer comando. Aqui te explicamos
↳ que comandos especiales tenemos disponibles y para que puedes
↳ usarlos\033[0m"
26             echo -e "\033[32mVuelve siempre que necesites ayuda con nuestros comandos
↳ especiales\033[0m"
27             ;;
28
29         "infosis")
30             echo -e "\033[33m\"infosis\" \033[32m Te mostraremos la información del
↳ sistema\033[0m"
31             echo -e "\033[32mLa memoria total, usada, libre, la arquitectura y la
↳ versión del SO\033[0m"
32             ;;
33
34         "fecha")
35             echo -e "\033[33m\"fecha\" \033[32m Verás la fecha y hora exacta al momento
↳ de ejecutarlo\033[0m"
36
37             ;;
38
39         "busca")
40             echo -e "\033[33m\"busca\" \033[32m Abrirá un menu con el cual puedes
↳ concontrar un archivo dentro del directorio que indiques. Debes ingresar
↳ la carpeta y tu archivo.\033[0m"

```

```

41         ;;
42
43     "creditos")
44         echo -e "\033[33m\"creditos \"\033[32m Verás a los grandiosos programadores
45         ↪ que crearon esta bonita terminal\033[0m"
46         ;;
47
48     "jugar")
49         echo -e "\033[33m\"jugar\"\033[32m Abrirá un juego clasico de gato (Tic Tac
50         ↪ Toe) muy entretenido para cuando estes aburrido\033[0m"
51         #Jugemos buscaminas o gato
52         ;;
53
54     "musica")
55         echo -e "\033[33m\"musica\"\033[32m Tendrás acceso a una biblioteca musical
56         ↪ para poder reproducir música\033[0m"
57         ;;
58
59     "salir")
60         echo -e "\033[33m\"salir\"\033[32m Huye de esta terminal, si tecleas esa
61         ↪ palabra mágica volverás a la normalidad\033[0m"
62         exit 0
63         #No olvides salir, esta es la unica oportunidad para no perderte en un bucle
64         ↪ infinito
65         ;;
66
67     *)
68         echo -e "\033[31m Comando no reconocido \033[0m"
69         ;;
70         esac
71
72 done

```



```

greta@/home/greta/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$
ayuda

Bienvenido.
Esta terminal cuenta con 8 comandos especiales para realizar divertidas actividades

===== MENÚ PRINCIPAL =====
Escribe el comando del cual requieras mas información
ayuda | infosis | fecha | busca | creditos | jugar | musica | salir
Dame info de > fecha

"fecha" Verás la fecha y hora exacta al momento de ejecutarlo

```

Figura 2: Uso del comando ayuda

1.1. Script infosis.sh

Este comando otorga información del sistema, tanto del software como del hardware. La información presentada es de la memoria ram, la arquitectura, y el sistema operativo utilizado por la computadora (o máquina virtual) donde se ejecuta el emulador. Para acceder a esta información, se accede a carpetas disponibles en muchos sistemas Linux, como en “proc” o en “etc”, en donde esta información se almacena en archivos, obteniendo con el comando grep y awk la información o las banderas exactas para poderla mostrar en pantalla.

- Memoria ram: Se obtiene del fichero `/proc/meminfo`, que guarda en kb la información de la memoria total y la memoria libre del sistema. Además, con una resta, se obtiene la memoria ocupada en ese momento.
- Arquitectura: Se busca una bandera llamada "lm" (long mode) en el fichero `/proc/cpuinfo` que indica que el sistema tiene 64 bits. En caso de no tenerla, el sistema trabaja con 32 bits.
- Sistema operativo: Busca variables almacenadas en el fichero `/etc/os-release` sobre el sistema operativo y la versión del computador.

```

1  #!/bin/bash
2
3  # Este script muestra información del sistema como la memoria, arquitectura y sistema
  ↪ operativo
4
5  # Obtenemr la información de la memoria ram
6  get_memory() {
7      # Leer el archivo /proc/meminfo para obtener información de la memoria
8      # AWK es una herramienta de procesamiento de texto que permite manipular y analizar
  ↪ datos
9      # grep busca la línea que contiene MemTotal y MemFree
10     # awk '{print $2}' extrae el segundo campo de la línea que contiene MemTotal y
  ↪ MemFree
11     total=$(grep MemTotal /proc/meminfo | awk '{print $2}')
12     free=$(grep MemFree /proc/meminfo | awk '{print $2}')
13     usada=$((total - free))
14     # Dividir por 1024 para convertir de kB a MB
15     echo "Total: $((total / 1024)) MB"
16     echo "Usada: $((usada / 1024)) MB"
17     echo "Libre: $((free / 1024)) MB"
18 }
19
20 # Obtener la arquitectura del sistema
21 get_architecture() {
22     # Busca si aparece "lm" (long mode) en las banderas (indicativo de 64 bits)
23     # grep busca la línea que contiene lm en /proc/cpuinfo
24     # Si se encuentra "lm", el sistema es de 64 bits, de lo contrario es de 32 bits
25     if grep -q "lm" /proc/cpuinfo; then
26         echo "Arquitectura: 64 bits"
27     else
28         echo "Arquitectura: 32 bits"
29     fi
30 }
31
32 # Obtener información del sistema operativo
33 get_os() {
34     # /etc/os-release es un fichero que contiene info sobre la distro del SO
35     . /etc/os-release
36     # NAME es el nombre de la distribución (declarada dentro del fichero)
37     # VERSION es la versión de la distribución (declarada dentro del fichero)
38     echo "Sistema Operativo: $NAME"
39     echo "Versión: $VERSION"
40 }
41
42 # Llamar a las funciones
43 echo "=== Información del Sistema ==="
44 get_memory

```

```
45 get_architecture
46 get_os
```

```
greta@/home/greta/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$
infosis
=== Información del Sistema ===
Total: 15868 MB
Usada: 9004 MB
Libre: 6864 MB
Arquitectura: 64 bits
Sistema Operativo: Ubuntu
Versión: 24.04.2 LTS (Noble Numbat)
```

Figura 3: Uso del comando infosis

1.2. Script fecha.sh

En este script, se ejecuta un comando personalizado que puede leer la fecha del sistema y mostrarla en pantalla. La restricción fue que no se podía usar el comando 'date' para obtener esta información, por lo que se recurrió al método de obtener el número de segundos usando el tiempo Unix, que es el tiempo transcurrido en segundos desde el 1 de enero de 1970, que funciona en todos los sistemas tipo Unix, variable conocida como "epoch". El programa hace la conversión a un formato legible para la fecha y la hora.

```
1  #!/bin/bash
2
3  #Todos los sistemas unix empiezan desde el 1 de enero de 1970
4  # y el tiempo se mide en segundos desde esa fecha (conocida como epoch)
5
6  #La opción '%(s)T' indica que se desea el tiempo en segundos desde epoch
7  epoch=$(printf '%(s)T\n' -1) # Obtener el número de segundos desde el 1 de enero de
   ↳ 1970 hasta el actual (-1)
8  fecha=$(printf '%(Y-%m-%d)T\n' "$epoch") #Convertimos segundos a un formato de fecha
   ↳ legible
9  hora=$(printf '%(H:%M:%S)T\n' "$epoch") # Convertimos segundos a un formato de hora
   ↳ legible
10
11 echo "La fecha es $fecha"
12 echo "La hora es $hora"
```

```
greta@/home/greta/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$
fecha
La fecha es 2025-04-26
La hora es 19:45:07
```

Figura 4: Uso del comando fecha

1.3. Script busca.sh

Este script te ayuda a buscar archivos que recuerdes haber guardado en alguna carpeta, recibe una dirección, recibe el nombre del archivo, verifica que la carpeta exista, si por defecto solo colocaste el nombre entonces busca en el directorio actual, también permite dar solo enter y ocupar la carpeta


```

1  #Para directorios no reconocidos mandamos un mensaje de error
2  #La bandera -d en un if comprueba si existe un directorio con el nombre $carpeta
3  if [ ! -d "$carpeta" ]; then
4      echo "La carpeta no existe"
5      exit 1
6  fi
7
8  echo "Buscando..... "
9  echo "Se encontraron los siguientes archivos"
10
11 find "$carpeta" -type f -iname "$archivo*"
12
13 #Busca con find archivos y directorios, en ruta $carpeta, seleccionamos solo archivos
14 ↪ con -type f
15 #Usamos -iname para buscar el nombre sin importar mayusculas y minusculas, y el uso de
16 ↪ asteriscos para ampliar la busqueda "$archivo*"

```

Figura 5: Enter Caption

[illegible]

[illegible]

```

15 tablero() {
16     echo "${board[0]} :: ${board[1]} :: ${board[2]}"
17     echo "::::::::::::"
18     echo "${board[3]} :: ${board[4]} :: ${board[5]}"
19     echo "::::::::::::"
20     echo "${board[6]} :: ${board[7]} :: ${board[8]}"
21 }
22
23 #Una funcion para el movimiento del cpu, es solo aleatorio
24 cpu(){
25     while true; do
26         pos=$((RANDOM % 9))
27         if [[ ${board[$pos]} == " " ]]; then
28             board[$pos]="o"
29             echo "La CPU eligio la posición $pos"
30             break
31         fi
32     done
33 }
34
35 #Con esta función podemos verificar si ya existe algun ganador con los movimientos
36 ↪ existentes
37 verificador() {
38
39     for i in 0 3 6; do
40         if [[ ${board[$i]} != " " && ${board[$i]} == ${board[$i+1]} && ${board[$i]} ==
41             ↪ ${board[$i+2]} ]]; then
42             echo ";Gana ${board[$i]}!"
43             tablero
44             exit
45         fi
46     done
47
48     for i in 0 1 2; do
49         if [[ ${board[$i]} != " " && ${board[$i]} == ${board[$i+3]} && ${board[$i]} ==
50             ↪ ${board[$i+6]} ]]; then
51             echo ";Gana ${board[$i]}!"
52             tablero
53             exit
54         fi
55     done
56
57     if [[ ${board[0]} != " " && ${board[0]} == ${board[4]} && ${board[0]} == ${board[8]}
58         ↪ ]]; then
59         echo ";Gana ${board[0]}!"
60         tablero
61         exit
62     fi
63
64     if [[ ${board[2]} != " " && ${board[2]} == ${board[4]} && ${board[2]} == ${board[6]}
65         ↪ ]]; then
66         echo ";Gana ${board[2]}!"
67         tablero
68         exit
69     fi
70 }

```

```

67
68 full=1
69 for cell in "${board[@]"; do
70     if [[ $cell == " " ]]; then
71         full=0
72         break
73     fi
74 done
75
76 if [[ $full -eq 1 ]]; then
77     echo "¡Empate!"
78     tablero
79     exit
80 fi
81
82
83 }
84
85 #Comienza el juego
86
87 while true; do
88     echo "Tu turno. Elige una posición (0-8): "
89     read pos
90
91     if [[ "$pos" == "salir" ]]; then
92         echo "Gracias por jugar. ¡Nos vemos luego!"
93         exit
94     fi
95
96     if [[ "$pos" =~ ^[0-8]$ && ${board[$pos]} == " " ]]; then
97         board[$pos]="X"
98         verificador
99         cpu
100        verificador
101        tablero
102    else
103        echo "Casilla inválida, intenta de nuevo"
104    fi
105
106 done
107

```

```

greta@/home/greta/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$
jugar
Bienvenido al famoso gato! Veamos si puedes vencerme
Escribe 'salir' en cualquier momento para terminar el juego
Tu turno. Elige una posición (0-8):
4
La CPU eligio la posición 7
  ::  ::
 ::: :::
  :: X ::
 ::: :::
  :: o ::
Tu turno. Elige una posición (0-8):
6
La CPU eligio la posición 5
  ::  ::
 ::: :::
  :: X :: o
 ::: :::
X :: o ::
Tu turno. Elige una posición (0-8):
2
¡Gana X!
  ::  :: X
 ::: :::
  :: X :: o
 ::: :::
X :: o ::
greta@/home/greta/Documentos/Proyecto_final_LINUX/Proyecto_final_linux_protecsa$

```

Figura 7: Uso del comando jugar, se repitió hasta ganar en 3 movimientos para esta captura

1.6. Script musica.sh

Para este comando necesitamos usar la paquetería mpg123 por lo que verificamos primero si la computadora lo tiene instalado, en caso contrario lo instalamos con sudo, el usuario ingresará su contraseña y se descargará automáticamente. Una vez obtenida pedimos que ingrese la ruta de la carpeta que quiere escuchar. Puede ser tan general como prefiera, seleccionaremos solo aquellos archivos que sea .mp3. Utilizamos whiptail como interfaz gráfica predeterminada, mostramos las canciones .mp3 encontradas, permitimos que se pueda seleccionar alguna y reproducir. La paquetería mpg123 incluye un menu interno cuando se esta reproduciendo la música, puedes acceder a el ingresando "h". Al terminar la canción que escogiste te pregunta si deseas reproducir otra. Si deseas salir antes de terminar la canción, ingresas "f"next track, y seleccionas «no>".

```

1  #!/bin/bash
2
3  # Verifica si mpg123 está instalado
4  if ! which mpg123 >/dev/null; then
5      echo -e "No tienes instalado el paquete mpg123, necesitas instalarlo para comenzar a
6      ↪ escuchar música"
7      read -p "¿Deseas instalarlo? (y/n): " input
8      if [[ "$input" == "y" ]]; then
9          sudo apt-get update && sudo apt-get install -y mpg123
10      else
11          exit
12      fi
13  fi

```

```

14 # Pide al usuario la ruta donde tiene guardada su música
15 read -p "Ingresa la ruta donde tienes guardada tu música: " ruta
16 if [[ ! -d "$ruta" ]]; then
17     echo "La ruta no existe"
18     exit 1
19 fi
20
21 # Función para mostrar lista de canciones
22 mostrar_lista_canciones() {
23     archivos=("$ruta"/*.mp3)
24     opciones=()
25     for archivo in "${archivos[@]"; do
26         nombre=$(basename "$archivo")
27         opciones+=("$archivo" "$nombre")
28     done
29
30     cancion=$(whiptail --title "Tu Biblioteca Musical" --menu "Elige una canción para
31 ↵ reproducir:" 20 78 10 "${opciones[@]}" 3>&1 1>&2 2>&3)
32
33     echo "$cancion"
34 }
35
36 # Función para reproducir canción
37 reproducir_cancion() {
38     cancion="$1"
39     clear
40     echo "Reproduciendo: $(basename "$cancion")"
41     mpg123 "$cancion"
42 }
43
44 # Comienza la navegación
45 while true; do
46     cancion=$(mostrar_lista_canciones)
47     if [[ -z "$cancion" ]]; then
48         break
49     fi
50
51     reproducir_cancion "$cancion"
52
53     if ! whiptail --yesno "¿Quieres reproducir otra canción?" 10 60; then
54         break
55     fi
56 done
57
58 clear
59 echo "Gracias por usar nuestro reproductor musical "

```

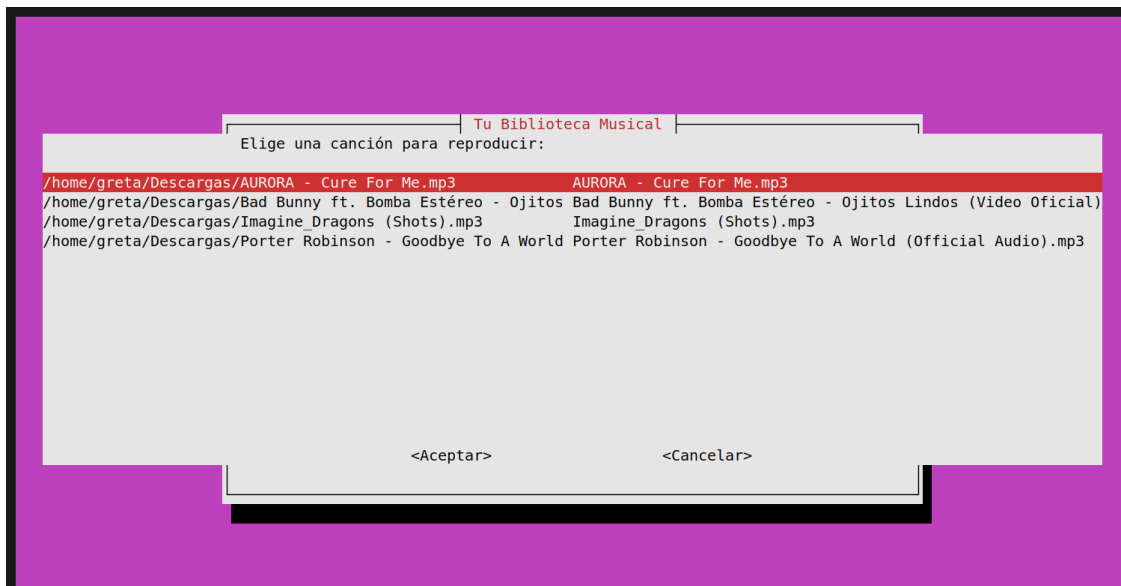


Figura 8: Interfaz gráfica del reproductor de música

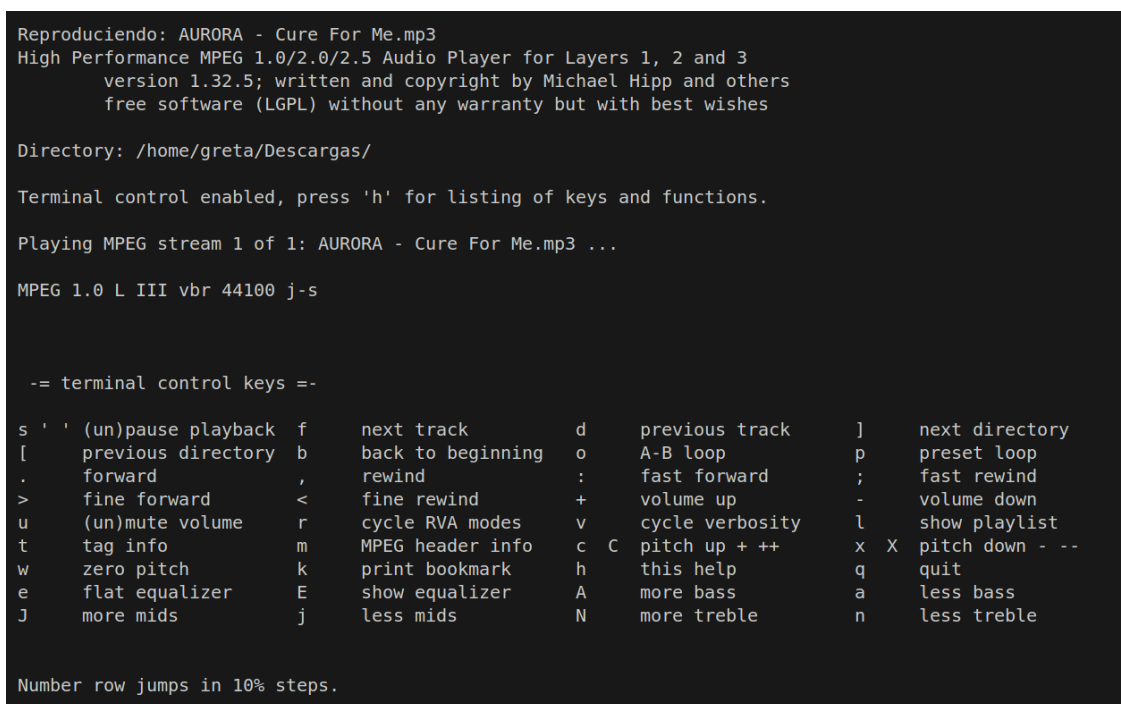


Figura 9: Menu interno de mpg123 usando la interfaz whiptail

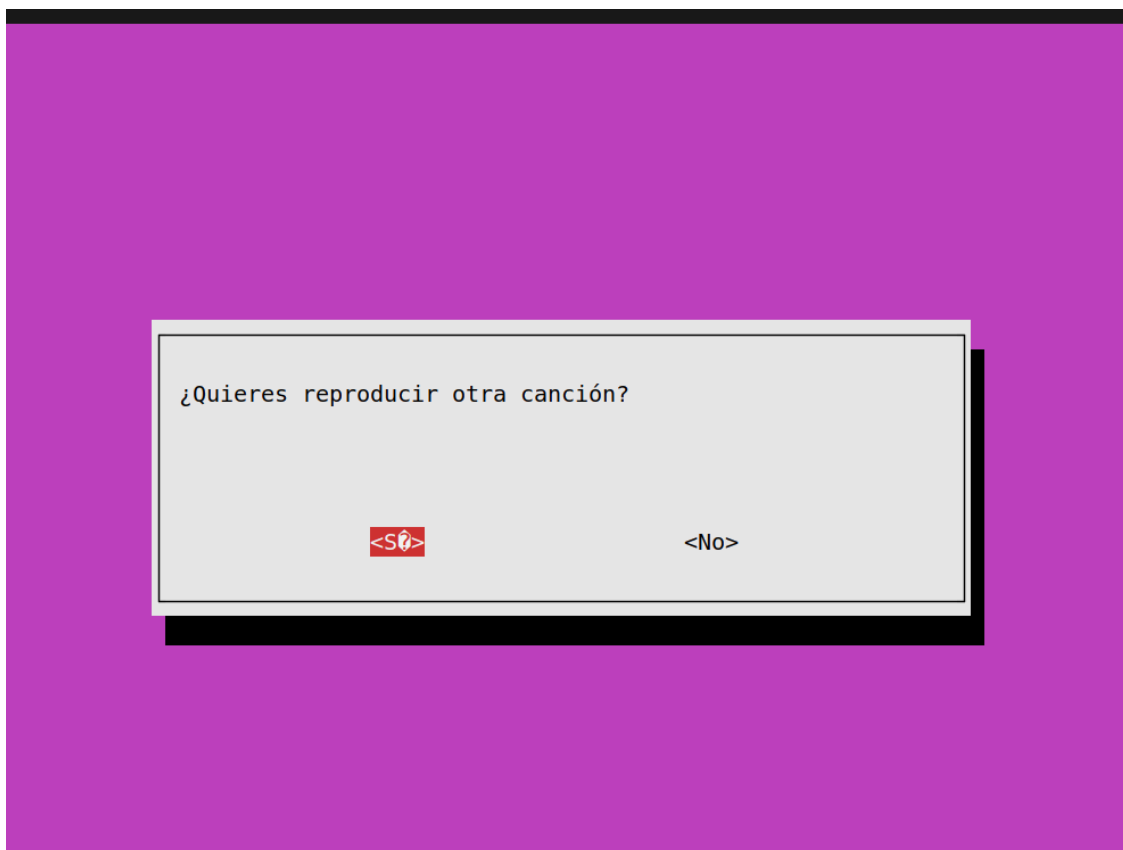


Figura 10: Menu para escuchar otra canción

2. Conclusiones por Buddy

2.1. Jair Montiel

La realización de un proyecto con estas características es fundamental para el desarrollo de habilidades, no sólo de programación, también de gestión de proyectos colaborativos. En un principio, se veía muy desafiante, en especial por las características tan particulares y las restricciones que se tuvieron en cuenta; sin embargo, se aplicaron exitosamente todos los comandos aprendidos durante el curso, que en retrospectiva, se utiliza lógica de programación que puede parecer trivial en algunos casos. Existieron otros como el del reproductor de mp3 que supusieron mayores retos, pero el trabajo colaborativo permitió el avance del proyecto. Además, se utilizaron metodologías ágiles para el desarrollo de software, cómo hacer reuniones diarias, establecer objetivos diarios y repartición equitativa de los programas. Cada uno de nosotros explicó detenidamente el código y la lógica de programación implementada, de manera que cada uno de nosotros pudiese entender lo que se estaba haciendo. Ambos aportamos al proyecto información y recomendaciones valiosas, que sumado al método de trabajo, permitió el rápido desarrollo del mismo.

2.2. Gretchen Rincón

Esta terminal emulada puede ser realmente útil como base para crear mini sistemas básicos en proyectos pequeños y caseros, o simplemente crear nuevas funciones para tu misma computadora, la creación de estos scripts nos dio la confianza de poder ejecutar nuevas instrucciones en el sistema, logramos implementar varias características, una dificultad que tuvimos fue como implementar los comandos sin que se parecieran del todo al comando original, por lo que optamos por ser más expresivos, sin embargo también tuvimos muchos detalles que quisiéramos mejorar, pues como todo proyecto siempre parece haber funciones y detalles que implementar tras cada commit y revisión hecha. El trabajo en equipo y el uso de Github potenció muy bien nuestro trabajo.