

UNIVERSIDAD NACIONAL AUTÓNOMA DE  
MÉXICO

FACULTAD DE INGENIERÍA



PROTECO  
GNU/Linux

---

## Terminal de trabajo PREBE

---

**Equipo 9:**

Rodríguez García Javier Antonio  
Treviño Selles Jorge Eithan

**GENERACIÓN 44**

**22 DE ABRIL DEL 2023**

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Desarrollo</b>	<b>3</b>
2.1. Sistema de acceso . . . . .	4
2.2. Línea de comandos . . . . .	8
2.3. Ayuda . . . . .	10
2.4. Información del sistema . . . . .	15
2.5. Fecha y hora . . . . .	17
2.6. Buscar . . . . .	18
2.7. Créditos . . . . .	19
2.8. Juego (ahorcado) . . . . .	21
2.9. Reproductor mp3 . . . . .	28
<b>3. Conclusiones</b>	<b>37</b>

# 1. Introducción

Durante este proyecto se elaboró una terminal de trabajo utilizando el lenguaje de programación Bash; esta terminal de trabajo es capaz de ejecutar los comandos dentro de un sistema Linux además de que posee algunos comandos adicionales para ejecutar programas elaborados por nosotros como lo es un juego o un reproductor mp3. Todos estos programas nos han ayudado tanto a aplicar los conocimientos vistos en el curso así como a investigar y pensar en cómo resolver ciertas particularidades tanto de Bash como de Linux (sintaxis, dónde se encuentran ubicados ciertos archivos, manejo de procesos simultáneos, etc).

El proyecto fue elaborado de forma colaborativa a lo largo de varios archivos con extensión .sh para así facilitar la modularidad y el trabajo en equipo. La colaboración se realizó a través de GitHub: utilizamos una convención para nombrar los commits además de establecer normativas para trabajar con las diferentes versiones del proyecto, esto para asegurar que los avances realizados fueran coherentes y mantener una forma de trabajo organizada y fácil de mantener.

# 2. Desarrollo

En esta sección describiremos brevemente el funcionamiento de cada uno de los componentes que elaboramos para esta terminal de trabajo. Cada uno de estos comandos se encuentra almacenado en diferentes archivos .sh cuyo nombre sin la extensión es el mismo que el archivo que lo ejecuta.

## 2.1. Sistema de acceso

```
javis@DESKTOP-GLLNMPF:/mnt/d/proyecto_final_protecto_linux-main/proyecto_final_protecto_linux-main$ ./login.sh
Ingrese su nombre de usuario:
jas
Usuario inválido
javis@DESKTOP-GLLNMPF:/mnt/d/proyecto_final_protecto_linux-main/proyecto_final_protecto_linux-main$ ./login.sh
Ingrese su nombre de usuario:
javis
Ingrese su contraseña:
Contraseña incorrecta
javis@DESKTOP-GLLNMPF:/mnt/d/proyecto_final_protecto_linux-main/proyecto_final_protecto_linux-main$ sudo useradd -s /bin/bash javier
[sudo] password for javis:
javis@DESKTOP-GLLNMPF:/mnt/d/proyecto_final_protecto_linux-main/proyecto_final_protecto_linux-main$ ./login.sh
Ingrese su nombre de usuario:
javier
passwd: You may not view or modify password information for javier.
La contraseña para el usuario javier está inactiva

BIENVENIDO

|| <<javis>> (proyecto_final_protecto_linux-main) ||: salir
javis@DESKTOP-GLLNMPF:/mnt/d/proyecto_final_protecto_linux-main/proyecto_final_protecto_linux-main$ ./login.sh
Ingrese su nombre de usuario:
javis
Ingrese su contraseña:
```

Como se

observa en la ejecución se pueden presentar 4 casos al iniciar sesión, en el primer caso, el usuario ingresado no existe en el sistema y por ende se devuelve un mensaje en el que se dice que usuario que no existe, en el segundo caso, el usuario existe, pero la contraseña ingresada no es correcta, el tercer caso se presenta cuando el usuario existe pero no cuenta con una contraseña, por último, el cuarto caso se presenta cuando las credenciales son válidas, una vez el usuario ingrese, aparecerá el mensaje de bienvenida, (en la captura se colocó de esa manera debido a que el logearse implica un clear, lo que borra la información y no es posible visualizar como es que se realizó el logeo).

### Explicación del código:

En el código se realizan 3 comparaciones, en la primera se comprueba que el usuario exista, esto lo hace mediante el comando `id`, el cual arroja información del usuario, por lo que si el usuario no existe, al arrojar un error, se activa el `if`, en ambos casos se cancela la salida que arroja el programa con el comando `exit /dev/null`, la segunda comparación se usa para saber si el usuario tiene contraseña, la tercera comparación se hace intentando logearse nuevamente, de esta manera, si se permite logear no se entra en el `if`, pero si no se permite logear, entra en el `if`, y se dice que la contraseña es incorrecta.

## Código:

---

```
1
2 #!/bin/bash
3
4 echo "Ingrese su nombre de usuario:"
5 read username
6
7 # Verificar si el usuario existe
8 if ! id "$username" &>/dev/null; then
9     echo "Usuario invlido"
10    return
11 fi
12
13 if [[ $(passwd -S "$username" | awk '{print $2}') == "P" ]]; then
14     echo "Ingrese su contraseña:"
15     read -s password
16     pass=1
17     if ! echo "$password" | su - "$username" -c 'echo ""' >/dev/null 2>&1; then
18         echo "Contrasea incorrecta"
19         pass=0
20     fi
21 else
22     echo "La contraseña para el usuario $username est inactiva"
23     pass=0
24 fi
25 if [ $pass -eq 1 ]
26 then
27     tput setaf 1
28     clear
29     echo "
30
31
32
33
34
35
36 "
37 sleep 0.2
38     tput setaf 2
39     clear
40     echo "
41
42
43
44
45
46 "
47     sleep 0.2
48     tput setaf 3
49     clear
50     echo "
```

```
51
52
53
54
55
56 "
57 sleep 0.2
58 tput setaf 4
59 clear
60 echo "
61
62
63
64
65
66 "
67 sleep 0.2
68 tput setaf 5
69 clear
70 echo "
71
72
73
74
75
76 "
77 sleep 0.2
78 tput setaf 6
79 clear
80 echo "
81
82
83
84
85
86 "
87 sleep 0.2
88 tput setaf 7
89 clear
90 echo "
91
92
93
94
95
96 "
97 sleep 0.2
98 tput setaf 8
99 clear
100 echo "
101
```

```
102
103
104
105
106 "
107     sleep 0.2
108     tput setaf 9
109     clear
110     echo "
111
112
113
114
115
116 "
117     sleep 0.2
118 clear
119 echo —e "\e[35m
120
121
122
123
124
125 \e[0m"
126
127 echo "
128     echo ""
129     echo ""
130 . ./linea—comando.sh
131 fi
```

---

## 2.2. Línea de comandos

```
|| <<nahtie>> (proyecto_final_protecto_linux) ||: █
```

Tras iniciar sesión, tendremos acceso a la línea de comando; dentro de ella podemos ingresar cualquier comando de Linux y se ejecutará como si se trata de la terminal Bash estándar además de poder también tener acceso a todos los comandos que hemos creado en este proyecto.

Dentro de la línea de comandos podremos ver el usuario que está utilizando el sistema actualmente así como el directorio donde se encuentra actualmente.

Un aspecto importante a considerar es que la principal forma para salir de esta terminal es utilizando la siguiente sentencia:

**Código:**

---

```
1 #!/bin/bash
2 # Overwrite the exit command
3 exit() {
4     :
5 }
6 # Overwrite the default trap for SIGTSTP, SIGINT and SIGTERM
7 trap '_ SIGTSTP SIGINT SIGTERM
8 while true
9 do
10     # Get current directory, and replace home directory with ~
11     DIR=$( basename "$PWD" | sed "s|$HOME|~|" )
12
13     # Show prompt with colors and read input
14     printf "\e[35m||<<\e[1;36m$USER\e[0m\e[35m>>_\e[36m$DIR\e[35m)_||:_\e[0m"
15     read input
16
17     # If command with the specified name exists inside the current directory, execute the script
18     if [ -e "$(echo_$input_|_cut_-d'_'_f1).sh" ]
19     then
20         if echo $input | grep -q " "; then
21             source "$_(echo_$input_|_cut_-d'_'_f1).sh" "$_(echo_$input_|_cut_-d'_'_f2)"
22         else
23             source "$input.sh"
24         fi
25     # If input is "salir", exit the loopd
26     elif [ "$input" == "salir" ]
27     then
28         break
29
30     # Execute OS command
31     else
```



```
32         $input
33     fi
34 done
35
36 # Return to the default trap for SIGTSTP, SIGINT and SIGTERM
37 trap -- SIGTSTP SIGINT SIGTERM
```

---

Los comandos *exit*, *CTRL+C*, *CTRL+X* y *CTRL+Z* están desactivados dentro de esta terminal.

### Explicación del código:

Primero sobrescribe las sentencias *exit*, *CTRL+C*, *CTRL+X* y *CTRL+Z* para prevenir que puedan ejecutarse. Posteriormente se entra en un ciclo eterno que muestra primero el usuario y directorio actual para después leer un string del usuario: primero se determinará si los caracteres antes del espacio de este string corresponden a uno de los comandos; en caso de que lo sea se determinará si este tiene un atributo para correr ya sea el script normal (agregando *.sh* a dicho comando y ejecutándolo) o dividir el string para una parte utilizarla para la ejecución y la otra para argumento. Si no se trata de un comando que hayamos realizado, se ejecutará directamente como si fuera una terminal de bash estándar. Al finalizar, se liberan las sentencias *exit*, *CTRL+C*, *CTRL+X* y *CTRL+Z* que habían sido reservadas

## 2.3. Ayuda

```
|| <<javis>> (proyecto_final_protecto_linux-main) ||: ayuda -a
COMANDOS DISPONIBLES

Comando      Descripción
----
infosis      Devuelve especificaciones del sistema, tales como:
              Memoria RAM, Arquitectura del sistema y
tiempo       Devuelve la hora y fecha local
buscar       Comando que sirve para buscar un archivo dentro de
              un directorio Especifico
creditos     Muestra los creditos de los programadores
ahorcado     ¿Necesitas un respiro? ¡JUEGA AHORCADO!
musica       Escucha Musica en nuestro nuevo reproductor
neko         ¿Quieres ver algo lindo?
salir        Salir de la línea de comandos
|| <<javis>> (proyecto_final_protecto_linux-main) ||:
|| <<javis>> (proyecto_final_protecto_linux-main) ||: ayuda -l
|| Comandos disponibles ||
-----
ahorcado
ayuda
buscar
creditos
infosis
musica
tiempo
salir
neko
|| <<javis>> (proyecto_final_protecto_linux-main) ||: ayuda infosis
|| Información del sistema ||
-----
- Uso:
  infosis
- Descripción:
  Muestra información del sistema:
    * Memoria RAM (total, libre y disponible).
    * Arquitectura
    * Sistema operativo y versión del kernel
|| <<javis>> (proyecto_final_protecto_linux-main) ||:
```

Este comando proporciona informacion de los comandos que se programaron para el proyecto, en este se tienen diferentes opciones, para poder utilizarlo se debe escribir *ayuda* seguido del comando del que se necesita la ayuda, ademas de que se pueden agregar las banderas *-a* para visualizar una descripcion general de todos los comandos y *-l* para visualizar unicamente los comandos que se pueden utilizar, sin ninguna descripción.

### Código:

```
1 #!/bin/bash
2 # Check if user has given an argument, if not, ask it
3 if [ -z $1 ] ; then
4     echo ""
5     read -p $"[35m->_[36mIngresa el comando del cual necesitas ayuda:_[0m'
6     command
7 else
8     command=$1
9 fi
10 if [ -z $command ] ; then
11     echo -e "[31m->_ERROR: No se ingres ningn comando"
```

```

12  # Check if command was created by us
13  case $command in
14      "—a" ) # Descripcin de todos los comandos
15          echo —e "\e[36mascii_art\e[0m"
16
17  printf "\n\n"
18  echo "Comando.....Descipcion"
19  printf "\n\n"
20  echo "infosis.....Devuelve especificaciones del sistema, tales como:"
21  echo ".....Memoria RAM, Arquitectura del sistema y"
22  printf "\n\n"
23  echo "tiempo.....Devuelve la hora y fecha local"
24  printf "\n\n"
25  echo "buscar.....Comando que sirve para buscar un archivo dentro de"
26  echo ".....un directorio Especifico"
27  printf "\n\n"
28  echo "creditos.....Muestra los creditos de los programadores"
29  printf "\n\n"
30  echo "ahorcado.....Necesitas un respiro? JUEGA AHORCADO!"
31  printf "\n\n"
32  echo "musica.....Escucha Musica en nuestro nuevo reproductor"
33  printf "\n\n"
34  echo "neko.....Quieres ver algo lindo?"
35  printf "\n\n"
36  echo "salir.....Salir de la linea de comandos"
37      ;;
38  "—l" ) # Lista de comandos disponibles
39      echo —e "\e[35m—————"
40      echo —e " _||_\e[36mComandos disponibles_\e[35m||"
41      echo —e " _—————\e[0m"
42      echo —e " _ahorcado"
43      echo —e " _ayuda"
44      echo —e " _buscar"
45      echo —e " _creditos"
46      echo —e " _infosis"
47      echo —e " _musica"
48      echo —e " _tiempo"
49      echo —e " _salir"
50      echo —e " _neko"
51      ;;
52  "ahorcado" ) # Juego ahorcado
53      echo —e "\e[35m—————"
54      echo —e " _||_\e[36mAhorcado_\e[35m||"
55      echo —e " _—————"
56      echo —e " —_\e[36mUso:_\e[0m"
57      echo —e " _ahorcado_\e[0m"
58      echo ""
59      echo —e " —_\e[36mDescripcin:_\e[0m"
60      echo —e " _Se ejecuta el juego del ahorcado: se trata de adivinar una
61  palabra secreta, ingresando letras. Si se adivina la palabra, se gana.
62  Si se adivina la palabra, se pierde."

```

```

63         ;;
64     "ayuda" ) # Comando de ayuda
65         echo -e "\e[35m_-----"
66         echo -e " _||_\e[36mComando_de_ayuda_\e[35m||"
67         echo -e " _-----"
68         echo -e " _\e[36mUso:_\e[0m"
69         echo -e " _ayuda_\e[0m"
70         echo -e " _ayuda_\e[1mcomando_a_consultar\e[0m"
71         echo -e " _ayuda_\e[1m-I\e[0m(ver_lista_de_comandos_disponibles))"
72         echo -e " _ayuda_\e[1m-a\e[0m(ver_descripcin_todos_los_comandos
73 _disponibles)"
74         echo ""
75         echo -e " _\e[36mDescripcin:_\e[0m"
76         echo -e " _Muestra_la_informacin_de_un_comando_creado_por_nosotros."
77         ;;
78     "buscar" ) # Comando de bsqueda
79         echo -e "\e[35m_-----"
80         echo -e " _||_\e[36mComando_de_bsqueda_\e[35m||"
81         echo -e " _-----"
82         echo -e " _\e[36mUso:_\e[0m"
83         echo -e " _buscar"
84         echo -e " _buscar_\e[1;36mdirectorio_donde_buscar\e[0m"
85         echo -e " _buscar_\e[1;36mdirectorio_donde_buscar_archivo_a_buscar\e[0m"
86         echo ""
87         echo -e " _\e[36mDescripcin:_\e[0m"
88         echo -e " _Busca_un_archivo_en_el_sistema_de_archivos:si_lo_encuentra
89 _muestra_un_mensaje_de_xito,si_no_lo_encuentra_muestra_que_no_lo
90 _encontr_y_si_el_directorio_no_existe_muestra_un_mensaje_de_error."
91         ;;
92     "creditos" ) # Comando para ver crditos de los autores
93         echo -e "\e[35m_-----"
94         echo -e " _||_\e[36mCrditos_de_los_autores_\e[35m||"
95         echo -e " _-----"
96         echo -e " _\e[36mUso:_\e[0m"
97         echo -e " _creditos_\e[0m"
98         echo ""
99         echo -e " _\e[36mDescripcin:_\e[0m"
100        echo -e " _Muestra_los_crditos_de_los_autores_del_proyecto_y_un
101 _personaje_de_Among_Us."
102        ;;
103    "infosis" ) # Comando para ver informacin del sistema
104        echo -e "\e[35m_-----"
105        echo -e " _||_\e[36mInformacin_del_sistema_\e[35m||"
106        echo -e " _-----"
107        echo -e " _\e[36mUso:_\e[0m"
108        echo -e " _infosis_\e[0m"
109        echo ""
110        echo -e " _\e[36mDescripcin:_\e[0m"
111        echo -e " _Muestra_informacin_del_sistema:
112 _*Memoria_RAM_(total,libre_y_disponible).
113 _*Arquitectura

```

```

114 .....*_Sistema_operativo_y_versin_del_kernel"
115         ;;
116     "musica" ) # Reproductor de mp3
117         echo -e "\e[35m_-----"
118         echo -e " _||_\e[36mReproductor_de_mp3_\e[35m||"
119         echo -e " _-----"
120         echo -e " _\e[36mUso:_\e[0m"
121         echo -e " _musica_\e[0m"
122         echo -e " _musica_\e[1mdirectorio_con_mp3\e[0m"
123         echo ""
124         echo -e " _\e[36mDescripcin:_\e[0m"
125         echo -e " _Abre_un_reproductor_de_msica_en_el_directorio
126 .....especificado_que_permite_reproducir_archivos_mp3._Si_no_se
127 .....especifica_un_directorio,_se_abre_en_el_directorio_actual."
128         ;;
129
130     "tiempo" ) # Ver tiempo actual
131         echo -e "\e[35m_-----"
132         echo -e " _||_\e[36mComando_para_ver_el_tiempo_\e[35m||"
133         echo -e " _-----"
134         echo -e " _\e[36mUso:_\e[0m"
135         echo -e " _tiempo_\e[0m"
136         echo ""
137         echo -e " _\e[36mDescripcin:_\e[0m"
138         echo -e " _Muestra_la_hora_y_fecha_actual."
139         ;;
140     "salir" ) #salir de la linea de
141         echo -e "\e[35m_-----"
142         echo -e " _||_\e[36mComando_para_finalizar_la_ejecucion_\e[35m||"
143         echo -e " _-----"
144         echo -e " _\e[36mUso:_\e[0m"
145         echo -e " _finalizar_ejecucion_del_programa_\e[0m"
146         echo ""
147         echo -e " _\e[36mDescripcin:_\e[0m"
148         echo -e " _Es_el_comando_unico_para_poder_finalizar_la_ejecucion_del
149 .....programa_y_salir_a_la_terminal_nativa_del_sistema."
150         ;;
151     "neko" ) #imprimer un neko
152         echo -e "\e[35m_-----"
153         echo -e " _||_\e[36mComando_para_ver_un_lindo_neko\e[35m||"
154         echo -e " _-----"
155         echo -e " _\e[36mUso:_\e[0m"
156         echo -e " _ver_un_lindo_gato,(se_recomienda_reducir_el_tamano_de_las
157 .....letras)_\e[0m"
158         echo ""
159         echo -e " _\e[36mDescripcin:_\e[0m"
160         echo -e " _La_tarea_o_el_trabajo_te_tienen_estresado?_Ve_un_lindo
161 .....gato_y_olvida_tus_problemas!"
162         ;;
163     * ) # Cualquier otro input
164         echo ""

```

```

165         echo -e "\e[35m->\e[31mLa informacin de este comando no est
166         disponible con ayuda""
167         echo -e "\e[35m->\e[36mPara ver la lista de comandos disponible,
168         ejecute \e[1;36mayuda -l\e[0m"
169         ;;
170     esac
171     echo ""
172 fi

```

---

## Sintaxis:

---

- 1 ayuda
  - 2 ayuda comando\_a\_consultar
- 

Para el funcionamiento del código se implementaron 2 if: el primero comprueba si ya se colocó un argumento al comando; si este argumento está vacío (es decir, en la terminal se colocó únicamente ayuda), le pide al usuario que introduzca el comando del que se quiere obtener la ayuda; el segundo if está colocado para comprobar que se haya colocado algún comando, en caso de que no se haya colocado se devolverá un mensaje de error; si se colocó algún comando se ingresará a una estructura *case* en la que dependiendo del comando que se escribió se devolverá la información correspondiente, en caso de colocar un comando que no exista se devolverá un mensaje informando al usuario.

## 2.4. Información del sistema



Al ejecutar este comando *infosis* se muestra en pantalla información acerca del sistema que estamos utilizando:

- **RAM**

Se muestra la memoria RAM total, utilizada y libre del sistema utilizando megabytes (MiB) como unidades.

- **Arquitectura**

Se muestra la arquitectura que utiliza nuestro sistema.

- **SO**

Se muestra el nombre de nuestro sistema operativo así como la versión del kernel que estamos utilizando.

### Código:

```
1 ## Cool logo
2 echo -e "\e[35m
3 -----
4 \e[36m
5
6
7
8
9
10 \e[35m-----"
```

```

11
12
13 ## RAM Usage
14 echo -e "\e[36m
15
16 "
17 # Get the memory usage in MiB and filter the output to only show the Mem row
18 mem_info=$(free -m | grep Mem)
19
20 # Extract the total used and free memory values as strings
21 total_mem=$(echo $mem_info | awk '{print $2}')
22 used_mem=$(echo $mem_info | awk '{print $3}')
23 free_mem=$(echo $mem_info | awk '{print $4}')
24
25 # Show memory values
26 echo -e "\e[1;35mTotal:\e[0m$total_mem_MiB"
27 echo -e "\e[1;35mUtilizada:\e[0m$used_mem_MiB"
28 echo -e "\e[1;35mLibre:\e[0m$free_mem_MiB"
29
30
31 ## System Architecture
32 echo -e "\e[35m-----\e[0m"
33
34 "
35 architecture=$(uname -m)
36 echo -e "
37 \e[1;35m$architecture\e[0m"
38
39 ## OS Version
40 echo -e "\e[35m-----\e[0m"
41
42 "
43
44 # Extract OS name and Kernel
45 os_name=$(sed -n 's/PRETTY_NAME="(.*\)" /\1/p' /etc/os-release)
46 kernel=$(uname -r)
47
48 # Show values
49 echo -e "\e[1;35mSistema operativo:\e[0m$os_name"
50 echo -e "\e[1;35mKernel:\e[0m$kernel"
51 echo -e "\e[35m-----\e[0m"
52 "

```

---

## Sintaxis:

---

```
1 infosis
```

---

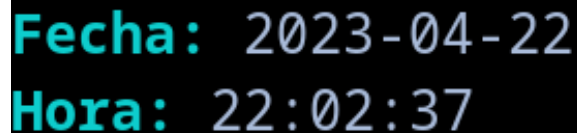
## Explicación del código:

Para obtener la memoria RAM se utiliza el comando `free` y después se muestra sólo la información necesaria (esta es extraída utilizando la herra-



mienta `awk`. La arquitectura se consigue utilizando el comando `$(uname -m)`. Finalmente, para obtener el nombre y kernel del sistema operativo utilizamos primero el programa `sed` para obtener el nombre de la distribución y después `$(uname -r)` para obtener el kernel. Toda esta información se muestra en pantalla.

## 2.5. Fecha y hora



```
Fecha: 2023-04-22
Hora: 22:02:37
```

Al ingresar el comando *tiempo* se mostrará en pantalla la fecha y hora del sistema al instante.

### Código:

---

```
1 printf "\n\e[1;36mFecha:\e[0m_"
2 cat /sys/class/rtc/rtc0/date
3 printf "\e[1;36mHora:\e[0m_"
4 cat /sys/class/rtc/rtc0/time
5 echo ""
```

---

### Sintaxis:

---

```
1 tiempo
```

---

### Explicación del código:

Imprime en pantalla el contenido de los archivos `/sys/class/rtc/rtc0/time` y `/sys/class/rtc/rtc0/date`.

## 2.6. Buscar

El comando buscar nos permite saber si un archivo se encuentra o no en un directorio especificado. Existen diferentes variantes de este programa ya que puede o no tomar argumentos: podemos especificar directamente en terminal el directorio y archivo a buscar o podemos también darlo de forma manual tras ejecutar el comando.

```
-> Ingrese el nombre del directorio dentro del que se va a buscar:
-> Ingrese el nombre del archivo a buscar:

EL ARCHIVO SI ESTA EN EL DIRECTORIO
```

Si el directorio no existe se le indicará al usuario en pantalla.

```
!!! NO EXISTE ESE DIRECTORIO !!!
```

El programa buscará dentro del directorio si se encuentra o no el archivo en dicho directorio le dirá al usuario si lo encontró o no.

```
EL ARCHIVO NO ESTA EN EL ESE DIRECTORIO
```

Código:

```
1 #!/bin/bash
2 function search()
3 {
4     if [ -d $1 ] ; then
5         if [[ -f "$1/$2" ]]; then
6             echo -e "\n\e[92m.....EL ARCHIVO SI ESTA EN EL DIRECTORIO"
7         else
8             echo -e "\n\e[33m.....EL ARCHIVO NO ESTA EN EL ESE DIRECTORIO"
9         fi
10    else
11        echo -e "\n\e[31m.....!!! NO EXISTE ESE DIRECTORIO !!!"
12    fi
13 }
14 if [ $1 ]; then
15     if [ $2 ]; then
16         search $1 $2
17     else
18         read -p $'\e[35m->\e[36mIngrese el nombre del archivo que se va a buscar:\e[0m'
19         dir
20         search $1 $dir
21     fi
22 fi
```

```

21 else
22     read -p $'\e[35m->\e[36mIngresa el nombre del directorio dentro del que se va a buscar:\e[0m'
    dir
23     read -p $'\e[35m->\e[36mIngresa el nombre del archivo a buscar:\e[0m'
    file
24     search $dir $file
25 fi
26 echo -e "\e[0m"

```

---

### Sintaxis:

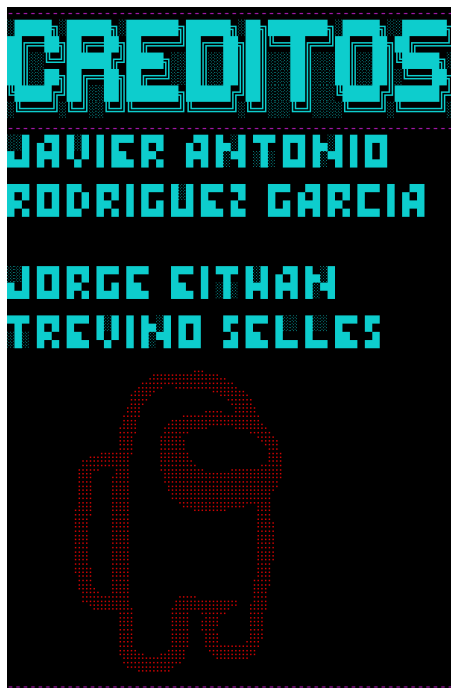
---

- 1 buscar
  - 2 buscar directorio
  - 3 buscar directorio archivo
- 

### Explicación del código:

Primero lee la entrada del usuario (ya sea que se le haya dado por terminal o pidiéndola directamente) y después averigua si se encuentra el archivo utilizando la bandera *-f* para después mostrar el mensaje en pantalla si es que lo encontró.

## 2.7. Créditos



Cuando ejecutamos el comando *credits* se mostrará un mensaje con los nombres de los autores de esta terminal seguido de una figura de Among Us.

## Código:

---

```
1 #!/bin/bash
2 echo -e "\e[35m
3 -----
4
5
6
7
8
9 \e[35m
10 -----
11 _
12 _
13 _
14 _
15 _
16 _
17 _
18 _
19 \e[31m
20 _
21 _
22 _
23 _
24 _
25 _
26 _
27 _
28 _
29 _
30 _
31 _
32 _
33 _
34 _
35 _
36 _
37 _
38 _\e[35m
39 -----
40 "
```

---

## Sintaxis:

1 creditos

## Explicación del código:

Muestra los nombres de los creadores y un Among Us en pantalla con ASCII art.

## 2.8. Juego (ahorcado)



El primero que aparece cuando se ejecuta el juego es un menu, en el que el usuario puede elegir la tema del que se adivinara la palabra, una vez elegido el tema se presentan 6 vidas y la interfaz del juego, dependiendo de si ganas o pierdas se presentaran dos interfaces, una felicitandote y otra en donde se le informara al usuario que se quedo sin vidas **Código:**

1 `#!/bin/bash`

2 `function dibujo {`

```

3   case $1 in
4   0)
5       echo " _____"
6       echo " |_____|"
7       echo " |_____|"
8       echo " |_____|"
9       echo " |_____|"
10      echo " |_____|"
11      echo " |_____|"
12      echo " |_____|"
13      echo " |_____|"
14      echo " |_____|"
15      echo " |_____|"
16      echo " |_____|"
17      echo " |_____|"
18      echo " |_____|"
19      ;;
20  1)
21      echo " _____"
22      echo " |_____|"
23      echo " |_____|"
24      echo " |_____|"
25      echo " |_____|"
26      echo " |_____|"
27      echo " |_____|"
28      echo " |_____|"
29      echo " |_____|"
30      echo " |_____|"
31      echo " |_____|"
32      echo " |_____|"
33      echo " |_____|"
34      echo " |_____|"
35      ;;
36  2)
37      echo " _____"
38      echo " |_____|"
39      echo " |_____|"
40      echo " |_____|"
41      echo " |_____|"
42      echo " |_____|"
43      echo " |_____|"
44      echo " |_____|"
45      echo " |_____|"
46      echo " |_____|"
47      echo " |_____|"
48      echo " |_____|"
49      echo " |_____|"
50      echo " |_____|"
51      ;;
52  3)
53      echo " _____"

```

```

54 echo " | "
55 echo " | "
56 echo " / \ "
57 echo " | "
58 echo " | "
59 echo " --- | "
60 echo " | "
61 echo " | "
62 echo " "
63 echo " "
64 echo " "
65 echo " | "
66 echo " "
67 ;;
68 4)
69 echo " "
70 echo " | "
71 echo " | "
72 echo " / \ "
73 echo " | "
74 echo " | "
75 echo " | "
76 echo " | "
77 echo " | "
78 echo " "
79 echo " "
80 echo " "
81 echo " | "
82 echo " "
83 ;;
84 5)
85 echo " "
86 echo " | "
87 echo " | "
88 echo " / \ "
89 echo " | "
90 echo " "
91 echo " "
92 echo " "
93 echo " "
94 echo " "
95 echo " "
96 echo " "
97 echo " | "
98 echo " "
99 ;;
100 6)
101 echo " "
102 echo " | "
103 echo " | "
104 echo " "

```

[illegible]



[illegible]



```

258                                     ;;
259     5)
260         juego=("${apellidos[@]}")
261         cat=("${nombrecat[4]}")
262         empezarJuego
263         ;;
264     *)
265         clear
266         echo adios
267         repetir=6
268         ;;
269     esac
270 done

```

---

## Sintaxis:

---

1 ahorcado

---

## Explicación del código:

Para poder elaborar el juego se declararon varios arreglos, estos contienen la información de la palabras con las que se jugara, posterior a esto se presenta un menu en el que se elige el tema, una vez elegido el tema, se manda llamar una funcion en la que se realiza el juego, los arreglos con las palabras y el tema se pasan como parametro a la funcion, en la funcion se elije un numero aleatorio del 0 al 19, debido a que hay 20 palabras, una vez elegida la palabra se mide su longitud y luego se genera un arreglo en el que se guardan varios -, el número de guiones bajos dependera del tamaño de la palabra, esto nos sirve para imprimir espacios vacios dentro del juego y para comprobar que haya letras por adivinar, después en un bucle while que compara el número de intentos sea diferente de 0, se realizan impreciones de pantalla propios del juego, tales como el numero de intentos(declarado al inicio de la funcion), el tema y el estado de el muñequito de ahorcado, este se imprime mandando llamar a otra funcion, en la que mediante una estructura case, se imprime un dibujo en funcion del numero de intentos restantes , ademas de el estado de la palabra(letras adivinadas y por adivinar), despues se le pide al usuario que ingrese una letra, posterior a esto, se compara la letra con cada una de las letras de la palabra para saber si es que la letra se encuentra en la palabra (esto se hace mediante un ciclo while), en caso de que se encuentre, esa letra sustituirá un espacio vacio en el arreglo, de tal manera que en la sigueite imprecision se visualize la letra adivianda, en este caso no se restan vidas. Si la letra no se encontro se restara una vida y por ende se imprimira un muñeco diferente, el juego se segeria ejecutando hasta que no haya espacios vacios o se acaben las vidas, en cualquiera de los casos, se regresará

al menu.

## 2.9. Reproductor mp3



Al ejecutar el comando *musica* tendremos acceso a un reproductor de archivos mp3 dentro del cual podemos reproducir canciones en un directorio específico. Este reproductor utiliza como base el programa mpg123, por lo que al ejecutarlo lo primero que hará es verificar que esté instalado y, si no lo está, lo instalará.

Si al ejecutar el programa no se le da un directorio como argumento, el programa lo solicitará con el siguiente mensaje:

```
Ingresa la ruta de dónde se encuentran las canciones(presiona enter si es en el directorio actual):
```

Si no se encuentra ningún archivo mp3, el programa simplemente se cerrará.

Dentro del menú. el usuario puede utilizar las diferentes teclas para navegar a través de su biblioteca, estas opciones son:

- Moverse entre canciones (flecha arriba/abajo)
- Reproducir la canción seleccionada
- Pausar la canción
- Continuar con la canción

- Saltar a la siguiente canción
- Regresar a la canción anterior
- Encender/Apagar el modo repetir
- Salir del programa

Dentro de la itnerfaz se puede ver el estado de reproducción (si está pausado o no), si está encendido el modo repetir y las canciones disponibles; si se está reproduciendo alguna canción se ilumina en verde. **Código:**

---

```

1  #!/bin/bash
2
3  # Verify mpg123 installation, if not installed, try to install it
4  if ! which mpg123 >/dev/null; then
5      echo -e "
6  ----->\e[31mADVERTENCIA:_NO_TIENES_INSTALADO_EL_PAQUETE_mpg123
7  -----\e[37mPara_ejecutar_esta_funcion_necesitas_instalarlo"
8      read -p "-----Deseas_instalarlo?(y/n):" input
9      if [ $input != "n" ] && [ $input != "N" ]
10     then
11         echo -e "
12  -----\e[37mDetectando_distribucion_de_Linux_para_instalar_la_dependencia..."
13         packagesNeeded='curl_jq'
14         if [ -x "$(command -v apk)" ]
15             then sudo apk add --no-cache mpg123
16         elif [ -x "$(command -v apt-get)" ]
17             then sudo apt-get install mpg123
18         elif [ -x "$(command -v dnf)" ]
19             then sudo dnf install mpg123
20         elif [ -x "$(command -v zypper)" ]
21             then sudo zypper install mpg123
22         elif [ -x "$(command -v pacman)" ]
23             then sudo pacman -S mpg123
24         else
25             echo -e "
26  ----->\e[31mERROR:_No_se_pudo_instalar_el_paquete_mpg123
27  -----\e[37mPara_continuar,_instala_manualmente_el_programa
28  -----\e[0m"
29             exit
30         fi
31     fi
32 fi
33
34 # Ask for playlist path if not already given
35 if [ -z "$1" ] || [ "$1" -eq "" ]; then
36     read -p $'
37 -----\e[37mIngresa_la_ruta_de_dnde_se_encuentran_las_canciones\e[2m(presiona_enter_si_es_en_el_directorio
38     if [ -z "$path" ] ; then # Assign current one if it's empty

```

```

39             path=$(pwd)
40         fi
41     else
42         path=$1
43     fi
44     if [[ $path != "/"* ]] ; then
45         path=$(pwd)/$path/
46     fi
47     if [ ! -d $path ] ; then
48         echo -e "
49         _____->\e[31mERROR: No existe el directorio
50         _____\e[37mCerrando programa...
51         _____\e[0m"
52         exit
53     fi
54
55     # Search for songs inside chosen path
56     i=0
57     songs=()
58     song_names=()
59     for file in $path*
60     do
61         if [ "${file##*.}" == "mp3" ] ; then
62             echo "Found song: $file"
63             # Add song to the array
64             songs+=("$file")
65             song_name="$(basename "$file")"
66             song_names+=("${song_name::-4}")
67             i+=1
68         fi
69     done
70     if [ $i == 0 ] ; then # No songs detected causes the program to close
71         echo -e "
72         _____->\e[31mERROR: No hay archivos .mp3 en el directorio
73         _____\e[37mCerrando programa...
74         _____\e[0m"
75     else
76
77         # Menu cycle
78         selected=0
79         current_song=-1
80         loop=false
81         pausa=false
82         pid=""
83         echo "" > tmp # tmp file to store name of the current song
84
85
86         # Play function
87         function play(){
88             # Play the selected song
89             while true ; do

```

```

90         echo "${song_names[$current_song]}" > tmp # Save the name of the current song
91         mpg123 -q "${songs[$current_song]}" # Play the song
92         # Increase counter
93         current_song=$((current_song+1))
94         # If loop is not enabled break the cycle, if not go back to the first song
95         if [ $current_song == $i ] ; then
96             if [ $loop == false ] ; then
97                 echo "" > tmp # Clear tmp file
98                 break
99             else
100                 current_song=0
101             fi
102         fi
103     done
104 }
105 # Kill track
106 function kill_track(){
107     # Kill the current song
108     if [ $current_song != -1 ] ; then
109         kill $pid
110     fi
111     pkill mpg123
112 }
113 while true ; do
114     # Clear the screen
115     clear
116
117     # Show controls
118     echo -e "\e[34m
119     """
120     """\e[35m\e[0m: Move entre las canciones
121     """\e[35mo\e[0m: Reproducir la cancin seleccionada
122     """\e[35mp\e[0m: Pausar la cancin
123     """\e[35mc\e[0m: Continuar la cancin
124     """\e[35ms\e[0m: Detener la cancin
125     """\e[35m\e[0m: Saltar a la siguiente cancin
126     """\e[35m\e[0m: Regresar a la cancin anterior
127     """\e[35mr\e[0m: Encender/Apagar el modo repetir
128     """\e[35mq\e[0m: Salir del programa
129     """
130
131     # Show songs title
132     echo -e "\e[34m
133     """
134     """
135
136     # Show playing status
137     if [ $current_song != -1 ] ; then
138         if [ $pausa == true ] ; then
139             echo -e "\e[31mPAUSADO"
140         else

```

```

141             echo -e "\e[32mREPRODUCIENDO"
142         fi
143     fi
144
145     # Show current loop status
146     if [ $loop == true ] ; then
147         echo -e "\e[94mREPETIR:_\e[32mON\e[0m"
148         "
149     else
150         echo -e "\e[94mREPETIR:_\e[31mOFF\e[0m"
151         "
152     fi
153
154     # Show available songs
155     for i in "${!song_names[@]}"; do
156         if [[ $i == $selected ]]; then
157             # Highlight the selected file with a ">" symbol
158             printf "\e[35m>_"
159         else
160             printf " _"
161         fi
162         # Compare current song with the one in the tmp file
163         if [ "${song_names[$i]}" == "${cat_tmp}" ] ; then
164             # Highlight the currently playing song in green
165             printf "\e[92m%s\n\e[0m" "${song_names[$i]}"
166         else
167             printf "\e[0m%s\n" "${song_names[$i]}"
168         fi
169     done
170
171     # Read a single character from the user
172     read -rsn1 -d '' input
173     # Execute actions based on the user input
174     case $input in
175         'q') # Quit
176             kill_track
177             rm tmp
178             clear
179             break
180             ;;
181         $'A') # Up arrow
182             if [ $selected -gt 0 ] ; then
183                 selected=$((selected - 1))
184             fi
185             ;;
186         $'B') # Down arrow
187             if [[ $selected -lt $i ]] ; then
188                 selected=$((selected + 1))
189             fi
190             ;;
191         # Play selected song

```



```

192             $'o')
193                 # Change status
194                 pausa=false
195             # Kill previous track
196             kill_track
197             # Play selected song
198             current_song=$selected
199                 play & pid=$!
200                 ;;
201         $'p') # Pause
202                 # Change status
203                 pausa=true
204                 kill -STOP $(pgrep mpg123) > /dev/null 2>&1
205                 ;;
206         $'c') # Continue
207                 # Change status
208                 pausa=false
209                 kill -CONT $(pgrep mpg123) > /dev/null 2>&1
210                 ;;
211         $'s') # Stop
212                 # Change status
213                 pausa=true
214                 # Kill previous track
215                 kill_track
216                 current_song=-1
217                 ;;
218         $'C') # Next
219                 # Change status
220                 pausa=false
221                 # Kill previous track
222                 kill_track
223                 # Find next song
224                 if [ $current_song -lt $i ] ; then
225                     current_song=$((current_song + 1))
226                     play & pid=$!
227                 elif [ $current_song -eq $i ] ; then
228                     if [ $loop == true ] ; then
229                         current_song=0
230                         play & pid=$!
231                     else
232                         current_song=-1
233                         pausa=true
234                         echo "" > tmp # Clear tmp file
235                     fi
236                 fi
237                 ;;
238         $'D') # Back
239                 # Change status
240                 pausa=false
241                 # Kill previous track
242                 kill_track

```

```

243         # Find previous song
244         if [ $current_song -eq -1 ] ; then
245             current_song=$i
246         elif [ $current_song -eq 0 ] ; then
247             if [ $loop == true ] ; then
248                 current_song=$i
249             else
250                 current_song=0
251             fi
252         else
253             current_song=$((current_song - 1))
254         fi
255         play & pid=$!
256         ;;
257     $'r') # Repeat
258         if [ $loop == false ] ; then
259             loop=true
260         else
261             loop=false
262         fi
263         ;;
264     esac
265 done
266 fi
267 reset

```

---

## Sintaxis:

- 
- 1 musica
  - 2 musica directorio\_con\_archivos\_mp3
- 

## Explicación del código:

Lo primero que realiza el programa es verificar que mpg123 esté instalado: si no lo está intentará instalarlo utilizando algunos de los diferentes manejadores de paquetes, si no lo encuentra se le dirá al usuario que no lo encontró y se le solicitará que lo haga para después cerrar el programa.

Tras cumplir con el requerimiento de tener mpg123 instalado, se verificará si el usuario dió o no un directorio al llamar al programa, si no lo dió se le solicitará que lo haga. Si el directorio no existe, se cerrará el programa. Si se pudo abrir el directorio exitosamente se recorrerán todos los archivos de este para ir almacenando los diferentes archivos mp3 que se encuentren; si no se encuentra ninguno el programa se cerrará.

Se inicializarán entonces algunas variables que nos ayudarán a llevar la cuenta de qué canción está seleccionada por el cursor, la canción en reproducción, si la canción está pausada, si está encendido el modo de repetición

y el pid del proceso que ejecutará las canciones, adicionalmente, se creará un archivo llamado *tmp* donde se almacenará la canción en reproducción; la razón por la que tuvimos que usar un archivo es porque la reproducción y el menú utilizan dos procesos distintos y la mejor forma que encontramos para que se comunicaran entre sí es mediante un archivo.

Entraremos entonces en un ciclo eterno donde primero se le muestran las opciones disponibles al usuario y luego la lista de canciones (incluyendo el cursor y la que se está reproduciendo), los estados del reproductor (si está en pausa y/o en repetición) para después esperar a que presione alguna tecla. Al presionar una tecla, puede realizarse lo siguiente dependiendo de cual sea:

- **q**  
Se detiene la reproducción de canciones, se borra el archivo temporal y se rompe el ciclo.
- **Tecla arriba**  
Se mueve el cursor hacia arriba.
- **Tecla abajo**  
Se mueve el cursor hacia abajo.
- **o**  
Se detiene la reproducción anterior, se cambia el estado de pausa a falso y se reproduce la canción actual en un proceso paralelo; el pid del proceso paralelo se almacena para que sea posible detener la reproducción en cuanto otra quiera tomar su lugar.
- **p**  
Pausa la reproducción y cambia el estado de pausa a verdadero.
- **c**  
Reanuda la reproducción y cambia el estado de pausa a falso.
- **s**  
Cancela la reproducción y cambia el estado de pausa a verdadero.
- **Tecla derecha**  
Se detiene la reproducción anterior, se cambia el estado de pausa a falso y se reproduce la canción siguiente a la que estaba; el pid del proceso paralelo se almacena para que sea posible detener la reproducción en cuanto otra quiera tomar su lugar. Si se trata de la última

canción se repetirá el ciclo si el modo de repetición está activado, de lo contrario, la reproducción se detendrá.

- **Tecla izquierda**

Se detiene la reproducción anterior, se cambia el estado de pausa a falso y se reproduce la canción anterior a la que estaba; el pid del proceso paralelo se almacena para que sea posible detener la reproducción en cuanto otra quiera tomar su lugar. Si se trata de la última canción se reproducirá la última canción si el modo de repetición está activado, de lo contrario, se repetirá la primera canción.

- **r**

Activa o desactiva el modo de repetición.

### 3. Conclusiones

- **Rodríguez García Javier Antonio**

A lo largo de la realización del programa, se me presentaron muchos percances, uno de ellos fue el inicio de sesión, pues la contraseña se encuentra cifrada, para evitar estos problemas, tuve que buscar muchas alternativas, sin embargo, al final conseguí los objetivos, en lo personal, el proyecto me ayudó mucho a mejorar mis capacidades de investigación y de programación en bash, además de contribuir en la mejora de mi lógica al programar, una de las cosas que más me gustaron fue la implementación del ascii art, pues me parece que darle un toque colorido al código hacer que se vea mejor y más elaborado, otra cosa que me llamó la atención, fue que en bash se pueden mandar llamar funciones con argumentos sin necesidad de declararlas, lo que facilitó muchas implementaciones.

- **Treviño Selles Jorge Eithan**

Durante este proyecto aprendí bastante y me familiaricé más con el lenguaje de programación Bash y los diferentes componentes del sistema operativo GNU/Linux. Realmente fue un proyecto que requirió de mucha investigación, tiempo y percepción cuando surgían errores. Afortunadamente el proyecto fue elaborado en equipos por lo que la carga de trabajo no fue tan alta, sin embargo, el trabajar en equipo también implica un balance y una comunicación clara de quién va a desarrollar qué parte y las diferentes convenciones a seguir. En general, puedo decir que fue un proyecto bastante didáctico e incluso desafiante en algunos puntos como lo fue el reproductor de música ya que ponía a prueba mis conocimientos de diferentes temas (sistemas operativos, estructuras de datos, etc).