UNIVERSIDAD DE COSTA RICA

ESCUELA DE INGENIERÍA ELÉCTRICA

IE0117 PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

Reporte

Laboratorio 2

Prof. Carolina Trejos Quirós

Estudiante: Diego Pereira[1], B85938

08 de setiembre de 2025

ÍNDICE

1. Introducción

Este informe de laboratorio requiere haber estudiado y practicado, independientemente del laboratorio anterior o no, para comprender cómo funcionan algunos aspectos introductorios referentes a la programación bajo plataformas abiertas y a la filosofía del software libre, especialmente GNU/Linux, y contar con herramientas básicas para interactuar con el Shell, específicamente Bash, que permite escribir código de algunos comandos en scripts que posteriormente se ejecutan desde alguna consola o terminal de comandos para obtener información o realizar labores específicas.

Previo a este laboratorio, se espera comprender el funcionamiento de algunas variables especiales en Bash, como lo son las variables de entorno o ambiente de usuario, y las variables especiales de entrada en scripting. También algunos argumentos de salida en scripting como lo es el redireccionamiento de la salida estándar (stdout), o de la salida de error (stderr). Además de conocer las estructuras de control, normalmente usadas para validar algunos datos de la entrada estándar (stdin), o en comparaciones de operadores numéricos, cadenas de caracteres, o de archivos.

Para este laboratorio, se espera entender que también existe el redireccionamiento de rutas, que por lo general es un archivo con un enlace simbólico (symbolic link) hacia otro archivo o directorio en otra ruta del sistema. También tener dominio en el control de usuarios y grupos, mediante la creación/modificación de cuentas y asignación/modificación de permisos de lectura/escritura/ejecución de archivos específicos. Y que también existe la posibilidad de, mediante las credenciales de otros usuarios del tipo administrador como lo es el root, ejecutar scripts más avanzados que requieren mayor nivel de permisos con los que no cuenta el usuario iniciado en la sesión actual.

2. Implementación

2.1. **Ejercicio1**: Scripting, usuarios y permisos

Dado que la asignación de permisos de archivo cuenta con 3 categorías diferentes, sean para el dueño, el grupo, u otros a veces llamados público (ni el dueño, ni ningún usuario perteneciente al grupo), se requiere saber cuál es el usuario y grupo al que el archivo tiene pertenencia. Cada categoría puede tener tanto, permiso de lectura, como de escritura, o de ejecución, lo que implica 9 subcategorías diferentes, además de la décima que sería el tipo que indica si es un archivo regular o directorio.

Debido a la diversidad de relaciones y permisos entre archivos, usuarios, y grupos, acá se muestra el mismo orden que sigue el script para validar cada acción, que evite en la mayor medida de lo posible identificadores huérfanos o relaciones rotas entre los mismos.

Antes de realizar alguna acción, primero se consulta o trata de obtener el ID o identificador de cada usuario y grupo, y viéndolo como una relación de menor a mayor, para evitar que al eliminar un grupo quede algún ID que los identifica sin relacionar, primero se consulta el ID de usuario, ya que es más probable que un usuario tenga menos relaciones con grupos distintos, que las que tiene un grupo con usuarios distintos.

2.1.1. Validar si existe un usuario

En caso de borrar alguno, primero se borra el usuario, y luego el grupo, por ello se decide consultar primero el usuario.

```
178         # linux-specific commands
179         # check if the user EXISTS
180         user_id=$(getent passwd | grep -A 0 -w -F "${OPT_USER_NAME}" |
            awk -F ':' '{print $3;}' 2>/dev/null)
181         if [[ -n "${user_id}" ]]; then
182             # user EXIST
```

Figura 1. Elaboración propia. *Comandos para obtener el identificador de un usuario*, 2025.

2.1.2. Validar si existe un grupo

Luego de borrar un usuario se puede borrar un grupo, sin embargo, ello no implica que el grupo tenga más relaciones de otros usuarios.

```
201         # check if the group EXISTS
202         group_id=$(getent group | grep -A 0 -w -F "${OPT_GROUP_NAME}" |
            awk -F ':' '{print $3;}' 2>/dev/null)
203         if [[ -n "${group_id}" ]]; then
204             # group EXIST
```

Figura 2. Elaboración propia. *Comandos para obtener el identificador de un grupo*, 2025.

2.1.3. Crear un grupo (si no existe)

Se crea primero el grupo, para que luego el usuario pueda contar con el identificador necesario para asociar la relación con el mismo.

```
221         # group DOES NOT exist
222         # creates the group
223         str_out=$(groupadd "${OPT_GROUP_NAME}" 1>/dev/null 2>&1)
224         group_id=$(getent group | grep -A 0 -w -F
            "${OPT_GROUP_NAME}" | awk -F ':' '{print $3;}' 2>/dev/null)
225         if [[ -n "${group_id}" ]]; then
```

Figura 3. Elaboración propia. *Comandos para crear un grupo y obtener su identificador*, 2025.

2.1.4. Crear un usuario (si no existe)

En este punto el grupo ya existe, por lo que, si se crea el usuario, se puede asignar la relación directa con el grupo, convirtiéndose en el grupo primario de dicho usuario. Sin embargo, si el usuario se crea sin especificar un grupo, la función por defecto es que crea un grupo con el mismo nombre que el usuario, y se asigna como el primario. En caso de asociar después otro grupo, pasaría a ser secundario, ya que sólo puede haber un primario, y muchos secundarios.

```
238         # check if the user DOES NOT exists
239         user_id=$(getent passwd | grep -A 0 -w -F "${OPT_USER_NAME}" |
            awk -F ':' '{print $3;}' 2>/dev/null)
240         if [[ -z "${user_id}" ]]; then
241             # user does NOT exist
242             # creates the user account
243             # -m, creates the user's home directory if it doesn't exist
244             # -s, default login to shell : "/bin/bash" | "/bin/zsh" |
                others
245             group_id=$(getent group | grep -A 0 -w -F
                "${OPT_GROUP_NAME}" | awk -F ':' '{print $3;}' 2>/dev/null)
246             if [[ -n "${group_id}" ]]; then # group EXISTS
247                 # BETTER IF SPECIFIES THE GROUP
248                 str_out=$(useradd -m -s "/bin/bash" "${OPT_USER_NAME}"
                    -g "${OPT_GROUP_NAME}" 1>/dev/null 2>&1)
249             else # group DOES NOT exist
250                 # AUTO CREATES A PRIMARY GROUP WITH THE SAME NAME. NOT
                    RECOMMENDED!!
251                 str_out=$(useradd -m -s "/bin/bash" "${OPT_USER_NAME}"
                    1>/dev/null 2>&1)
252             fi
253             # sets the user password
254             str_out=$(passwd "${OPT_USER_NAME}")
255             user_id=$(getent passwd | grep -A 0 -w -F
                "${OPT_USER_NAME}" | awk -F ':' '{print $3;}' 2>/dev/null)
256             if [[ -n "${user_id}" ]]; then
```

Figura 4. Elaboración propia. *Comandos para crear un usuario y establecer una contraseña*, 2025.

2.1.5. Agregar un usuario a un grupo

Lo normal es que este tipo de asignación cree únicamente relaciones secundarias, ya que, en la mayoría de los casos, el usuario ya va a tener un grupo primario asociado.

```
270         # check if user belongs to group
271         str_out=$(groups "${OPT_USER_NAME}" | grep -A 0 -w -F
            "${OPT_GROUP_NAME}" 2>/dev/null)
272         if [[ -z "${str_out}" ]]; then
273             # user does not belong to group
274             # to grant the user additional permissions
275             # can add users to existing groups like
276             # root for administrative privileges
277             str_out=$(gpasswd -a "${OPT_USER_NAME}" "${OPT_GROUP_NAME}"
                2>&1)
278             str_out=$(echo "${str_out}" | sed "s|Adding user
                ${OPT_USER_NAME} to group ${OPT_GROUP_NAME}||g")
279             str_out=$(usermod -aG "${OPT_GROUP_NAME}"
                "${OPT_USER_NAME}" 2>&1)
280             if [[ -z "${str_out}" ]]; then
```

Figura 5. Elaboración propia. *Verificar y agregar la pertenencia de un usuario a un grupo*, 2025.

2.1.6. Modificar los permisos de un archivo al usuario y grupo

Aunque un usuario y/o grupo aparezcan como propietarios del archivo, no necesariamente ambos pueden tener los mismos permisos para dicho archivo, ya que cada una de las 3 categorías incluyendo la pública, pueden tener permisos diferentes.

```
331         str_out=$(chown -cRHL "${OPT_USER_NAME}:${OPT_GROUP_NAME}"
            "${FILE_INPUT_STR}" 2>/dev/null)
332         if [[ $? -eq 0 ]]; then
333             if [[ -z "${str_out}" ]]; then
334                 printgreen "Usuario y grupo ya son dueños del archivo"
335                 echo
336             else
337                 printgreen "Se modifica la pertenencia del archivo"
338                 echo
339             fi
340         else
341             printred "No se pudo modificar la pertenencia del archivo"
342             echo
343         fi
344         str_out=$(chmod -cRHL 740 "${FILE_INPUT_STR}" 2>/dev/null)
345         if [[ $? -eq 0 ]]; then
346             if [[ -z "${str_out}" ]]; then
347                 printgreen "Usuario y grupo ya tienen los permisos"
348                 echo
349             else
350                 printgreen "Se modifican los permisos del archivo"
351                 echo
352             fi
353         else
354             printred "No se pudo modificar los permisos del archivo"
355             echo
356         fi
```

Figura 6. Elaboración propia. *Comandos para modificar el dueño de un archivo y sus permisos*, 2025.

2.2. **Ejercicio2**: Scripting y procesos

Existen los procesos del sistema u otros que se crean al ejecutar un comando para realizar labores específicas. Dichos procesos pueden tener diferente tipo de estado, y aunque el sistema operativo cuenta con sus propios métodos para la administración de los mismos, y liberar recursos no utilizados, siempre cabe la posibilidad que alguno quede sin ser liberado, por lo que muchas veces la única forma es liberarlo manualmente. Para dicha gestión hay herramientas útiles, tanto para monitorear como para finalizar procesos.

2.3. **Ejercicio3**: Scripting y servicios

Los servicios existen precisamente para administrar procesos, por lo que se puede encontrar servicios para procesos que no dependen tanto de la sesión de un usuario, sino que son utilizados de forma general por el sistema para realizar gestiones específicas.

2.3.1. Servicio para monitorear los cambios de un directorio

Este script es usado por el servicio que se habilita en el sistema para ejecución en background.

```
82      # be sure that output file exists
83      # be care of not saving on same watching directory
84      # because the modify event is trigger indefinitely
85      # and the output file size will increases a lot
86      touch ${file}
87      # format for the timestamp in the output
88      # --timefmt cannot be specified without --format
89      local timefmt="%Y-%m-%d_%H.%M.%S"
90      # -c and --format cannot both be specified
91      # %T, formatted timestamp
92      # %e, event names (CREATE, MODIFY, DELETE)
93      # %w, watched file or directory
94      # %f, filename that caused the event
95      local format=""
96      # define the type of events to watch for
97      local events="create,modify,delete"
98      # BACKGROUND: DAEMON WITH OUTPUT FILE (CSV)
99      # DOES NOT WORK TO START THE SERVICE
100     #format="\"%T\",\"%_e\",\"%w\",\"%f\""
101     #inotifywait -drs -o "${file}" --timefmt "${timefmt}" --format
        "${format}" -e "${events}" "${dir}" | while read -r timestr nbsp
        events nbsp path nbsp filename
102     # FOREGROUND: MONITOR WITH OUTPUT FILE (CSV)
103     # USING THIS TO START THE SERVICE
104     format="\"%T\",\"%_e\",\"%w\",\"%f\""
105     inotifywait -mrs -o "${file}" --timefmt "${timefmt}" --format
        "${format}" -e "${events}" "${dir}" | while read -r timestr nbsp
        events nbsp path nbsp filename
106     # FOREGROUND: MONITOR IN TERMINAL (REALTIME)
107     # USING THIS TO SEE IN TERMINAL
108     #format="\"%T\" , \"%_e\" , \"%w\" , \"%f\""
109     #inotifywait -mrs --timefmt "${timefmt}" --format "${format}" -e
        "${events}" "${dir}" | while read -r timestr nbsp events nbsp path
        nbsp filename
110     do
```

Figura 10. Elaboración propia. *Servicio para monitorear eventos, hacia archivo o pantalla*, 2025.

2.3.2. Script que inicia el servicio de monitoreo en vivo

Este script requiere ser ejecutado con permisos de un administrador como el root (sudo).

```
# inotifywait is a command-line utility from the inotify-tools package
# so needs to install inotify-tools
# Arch Linux
#pacman -S inotify-tools
# Debian Ubuntu Mint
#apt install inotify-tools
# Red Hat CentOS Fedora
#yum install inotify-tools
# Fedora RHEL newer versions
#dnf install inotify-tools
# openSUSE
#zypper install inotify-tools

cp -fRHL ejercicio3daemon /tmp/
#cp -fRHL ejercicio3.service /etc/systemd/system/
cp -fRHL ejercicio3.service /usr/lib/systemd/system/
ln -s /usr/lib/systemd/system/ejercicio3.service /etc/systemd/system/
ejercicio3.service
systemctl daemon-reload
#systemctl start ejercicio3.service
systemctl restart ejercicio3
exit 0
```

Figura 11. Elaboración propia. *Comandos necesarios para iniciar el servicio de monitoreo,* 2025.

2.3.3. Script que finaliza el servicio de monitoreo en vivo

Este script requiere ser ejecutado con permisos de un administrador como el root (sudo).

```
systemctl --kill-whom=all --signal=SIGTERM kill ejercicio3
# does not finishes older instances, only the current
systemctl stop ejercicio3
# terminates all instances of process
# -f, matches only the full name, not only containing the word as regex
# -n, newest
# -o, oldest
#pkill -f "inotifywait"
# killall is an alternative to pkill
killall "inotifywait"
# prints if there are instances running
# -n, newest
# -o, oldest
#pgrep -n "inotifywait"
# prints only the ID of all instances
#pgrep "inotifywait"
# ps and grep is an alternative to pgrep
# -a, matches all
# -u, current user
# -x, must have a tty
ps aux | grep "inotifywait" | grep -v grep
exit 0
```

Figura 12. Elaboración propia. *Comandos necesarios para detener el servicio de monitoreo,* 2025.

3. RESULTADOS

3.1. Ejercicio1: Scripting, usuarios y permisos

```
[diego@pereira Scripts]$ ~/ej1 Lab02~! IE0117
Usuario de la sesion: diego
Faltan datos en los argumentos del script.
```

Figura 13. Elaboración propia. *Validar la cantidad de argumentos necesarios*, 2025.

```
[diego@pereira Scripts]$ ~/ej1 Lab02~! IE0117 ~/file2
Usuario de la sesion: diego
Usuario solo acepta letras, numeros, guion -, y guion bajo _
```

Figura 14. Elaboración propia. *Validar el formato de un nombre de usuario*, 2025.

```
[diego@pereira Scripts]$ ~/ej1 Lab02 IE0117~! ~/file2
Usuario de la sesion: diego
Grupo solo acepta letras, numeros, guion -, y guion bajo _
```

Figura 15. Elaboración propia. *Validar el formato de un nombre de grupo*, 2025.

```
[diego@pereira Scripts]$ ~/ej1 Lab02 IE0117 ~/file2 ~/file11
Usuario de la sesion: diego
No existe ningun archivo indicado en los argumentos
AYUDA
Uso del comando, ej.: ejercicio1 usuario grupo [/ruta/]archivo
NOMBRE:
        ejercicio1 asigna a usuario y grupo permisos del archivo.
OPCIONES:
        usuario
        Usuario que requiere permisos para el archivo.
        grupo
        Grupo al que se asigna permisos y el usuario.
        archivo
        Archivo del sistema que requiere permisos.
[diego@pereira Scripts]$
```

Figura 16. Elaboración propia. *Validar la existencia de un archivo para permisos*, 2025.

```
[diego@pereira Scripts]$ ls -lah
total 24K
drwxrwxrwx 1 diego diego  20 Sep 11 09:27
drwx------ 1 diego diego 426 Sep 11 05:13 ..
-rwxr-xr-x 1 diego diego 23K Sep 11 09:31 ejercicio1
[diego@pereira Scripts]$ ~/ej1 Lab02 IE0117 ~/file1
Usuario de la sesion: diego
Usuario que ejecuta : diego
El comando ingresado: /home/diego/ej1 Lab02 IE0117 /home/diego/file1
El comando procesado: /home/diego/Scripts/ejercicio1 Lab02 IE0117 /home
/diego/Scripts2/ejercicio1
Este script require permisos de root
[sudo] password for diego:
```

Figura 17. Elaboración propia. *Validar la ejecución con permisos de administrador*, 2025.

```
Usuario de la sesion: diego
Usuario que ejecuta : root
El comando ingresado: /home/diego/ej1 Lab02 IE0117 /home/diego/file2 /h
ome/diego/file1
El comando procesado: /home/diego/Scripts/ejercicio1 Lab02 IE0117 /home
/diego/Scripts2/ejercicio1
Se crea el grupo IE0117 con ID 1001
New password:
Retype new password:
passwd: password updated successfully
Se crea el usuario Lab02 con ID 1001
Usuario Lab02 ya pertenece al grupo IE0117
ACERCA DEL ARCHIVO /home/diego/Scripts2/ejercicio1
Para el usuario Lab02 y grupo IE0117
Se modifica la pertenencia del archivo
Se modifican los permisos del archivo
[diego@pereira Scripts]$
```

Figura 18. Elaboración propia. *Crear grupo, usuario, y asignar permisos de archivo*, 2025.

```
Usuario de la sesion: diego
Usuario que ejecuta : root
El comando ingresado: /home/diego/ej1 Lab02 IE0117 /home/diego/file2 /h
ome/diego/file1
El comando procesado: /home/diego/Scripts/ejercicio1 Lab02 IE0117 /home
/diego/Scripts2/ejercicio1
Usuario Lab02 ya existe con ID 1001
Grupo IE0117 ya existe con ID 1001
Usuario Lab02 ya pertenece al grupo IE0117
ACERCA DEL ARCHIVO /home/diego/Scripts2/ejercicio1
Para el usuario Lab02 y grupo IE0117
Usuario y grupo ya son dueños del archivo
Usuario y grupo ya tienen los permisos
[diego@pereira Scripts]$
```

Figura 19. Elaboración propia. *Validar si existen permisos de archivo, usuario, y grupo*, 2025.

```
[diego@pereira Desktop]$ ls -lah ../Scripts2
total 24K
drwxrwxrwx 1 diego diego  20 Sep 11 09:45 .
drwx------ 1 diego diego 426 Sep 11 05:13 ..
-rwxr-xr-x 1 diego diego 23K Sep 11 09:45 ejercicio1
[diego@pereira Desktop]$ ls -lah ../Scripts2
total 24K
drwxrwxrwx 1 diego diego   20 Sep 11 09:45 .
drwx------ 1 diego diego  426 Sep 11 05:13 ..
-rwxr----- 1 Lab02 IE0117 23K Sep 11 09:45 ejercicio1
[diego@pereira Desktop]$
```

Figura 20. Elaboración propia. *Modificar la pertenencia y los permisos de un archivo*, 2025.

3.2. **Ejercicio3**: Scripting y servicios

```
[diego@pereira Scripts]$ ls -lah
total 28K
drwxrwxrwx 1 diego diego  160 Sep 11 09:02 .
drwx------ 1 diego diego  426 Sep 11 05:13 ..
-rwxr-xr-x 1 root  root  7.8K Sep 11 09:01 ejercicio3daemon
-rwxr-xr-x 1 root  root  7.9K Sep 11 09:01 ejercicio3monitor
-rwxr-xr-x 1 root  root   624 Sep 11 09:01 ejercicio3.service
-rwxr-xr-x 1 root  root   478 Sep 11 09:01 ejercicio3start
-rwxr-xr-x 1 root  root  1.6K Sep 11 09:01 ejercicio3stop
[diego@pereira Scripts]$ sudo ./ejercicio3start
[sudo] password for diego:
ln: failed to create symbolic link '/etc/systemd/system/ejercicio3.serv
ice': File exists
[diego@pereira Scripts]$ sudo ./ejercicio3stop
[sudo] password for diego:
inotifywait: no process found
[diego@pereira Scripts]$
```

Figura 31. Elaboración propia. *Scripts para iniciar o detener el servicio de monitoreo*, 2025.

```
[diego@pereira tmp]$ systemctl status ejercicio3
● ejercicio3.service - Servicio de Monitoreo de Cambios en Directorios
     Loaded: loaded (/usr/lib/systemd/system/ejercicio3.service; bad; prese
t: disabled)
     Active: active (running) since Thu 2025-09-11 09:07:13 CST; 2min 29s a
go
 Invocation: 3d5b85ab9ce34c739b4e116e316e9c20
   Main PID: 13629 (ejercicio3daemo)
      Tasks: 2 (limit: 4625)
     Memory: 1.5M (peak: 2.9M)
        CPU: 136ms
     CGroup: /system.slice/ejercicio3.service
             ├─13629 /bin/bash /tmp/ejercicio3daemon # Command to start the
 service
             └─13646 inotifywait -mrs -o /var/tmp/inotify_events.csv --time
fmt %Y-%m-%d_%H.%M.%S --format "\"%T\",\"%_e\",\"%w\",\"%f\"" -e create,mod
ify,>

Sep 11 09:07:13 pereira systemd[1]: Started Servicio de Monitoreo de Cambio
s en Directorios.
Sep 11 09:07:13 pereira ejercicio3daemon[13632]: logname: no login name
Sep 11 09:07:13 pereira ejercicio3daemon[13629]: Usuario de la sesion:
Sep 11 09:07:13 pereira ejercicio3daemon[13629]: Usuario que ejecuta : root
Sep 11 09:07:13 pereira ejercicio3daemon[13629]: El comando ingresado: /tmp
/ejercicio3daemon # Command to start the service
Sep 11 09:07:13 pereira ejercicio3daemon[13629]: El comando procesado: /tmp
/ejercicio3daemon
Sep 11 09:07:13 pereira inotifywait[13646]: Setting up watches.  Beware: si
nce -r was given, this may take a while!
Sep 11 09:07:13 pereira inotifywait[13646]: Watches established.
[diego@pereira tmp]$
```

Figura 32. Elaboración propia. *Estado del servicio, activo (running) o detenido (dead)*, 2025.

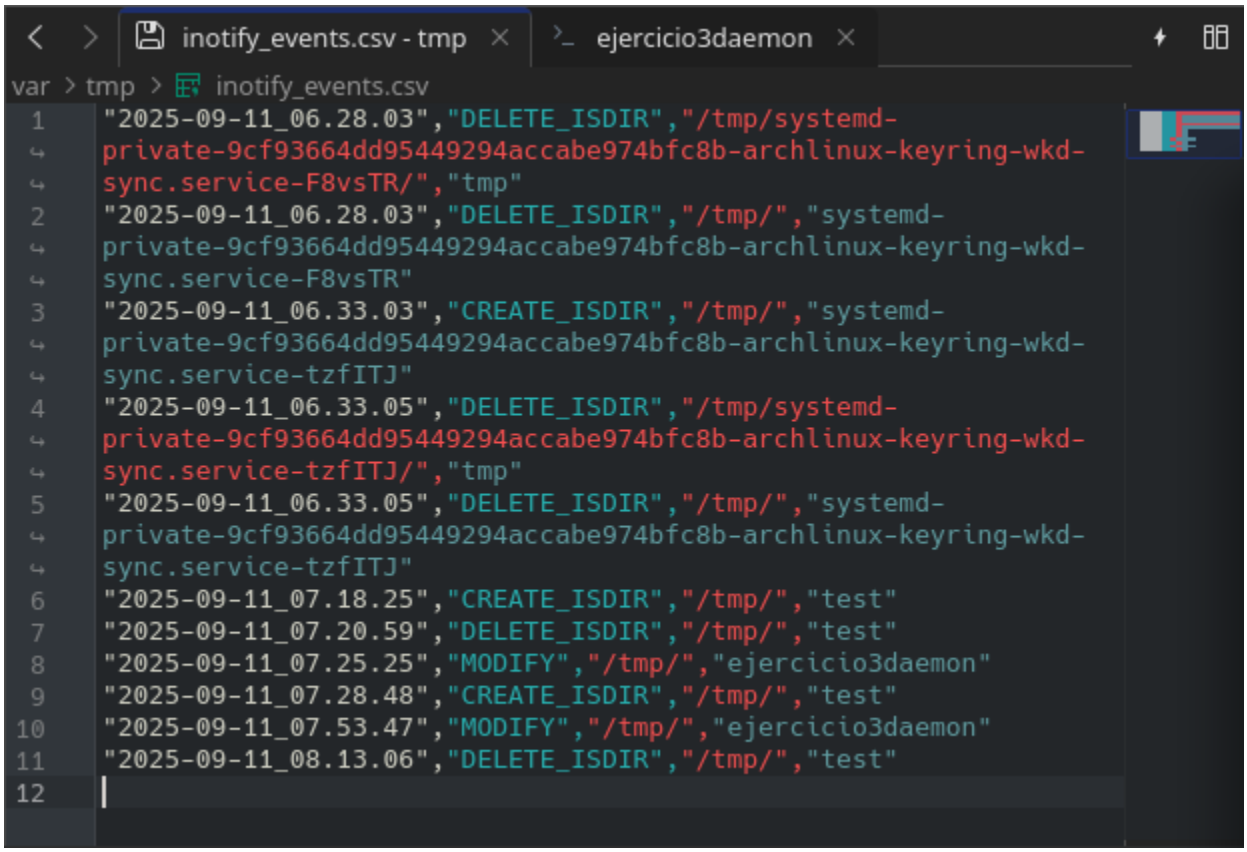

Figura 33. Elaboración propia. *Monitoreo de eventos hacia un archivo (background),* 2025.

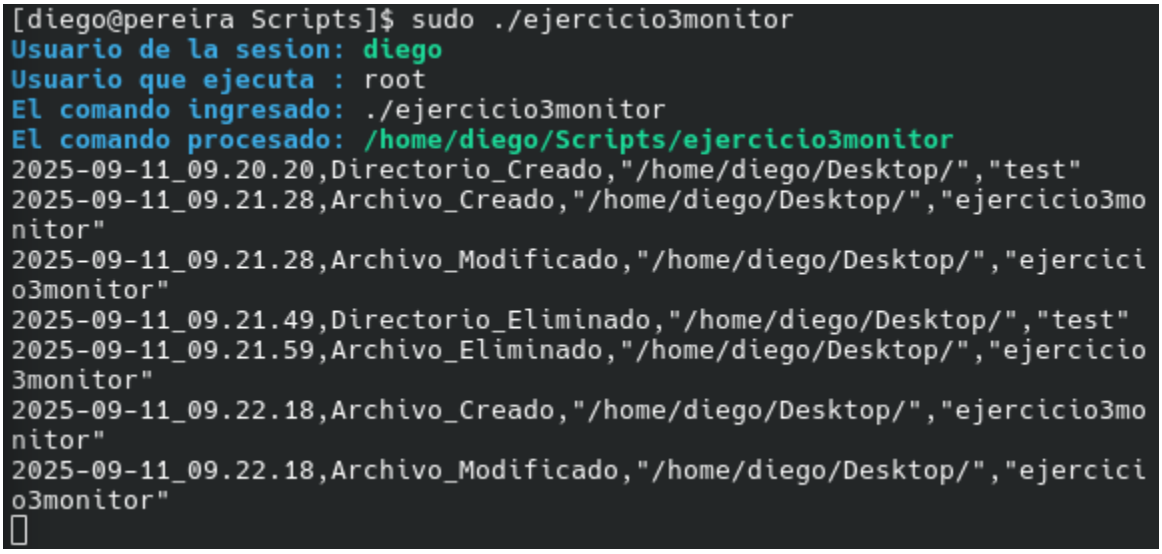

Figura 34. Elaboración propia. *Monitoreo de eventos hacia la pantalla (realtime),* 2025.

4. Análisis de Resultados

Para la sección 3.1 acerca de usuarios y permisos, se puede argumentar de las pruebas realizadas tanto en distribuciones GNU (requisito) de Arch Linux y Linux Mint, como non-GNU (macOS 10.14), donde ambos tipos cuentan con un Shell en Bash y la habilidad para ejecutar comandos similares. Se obtienen resultados iguales para los comandos de mismo nombre, por lo que cabe recalcar, que para algunas acciones específicas sí hay que validar y especificar mediante el código la diferencia entre los comandos Bash de un sistema GNU vs non-GNU.

En ambos sistemas se obtienen resultados similares, al poder crear usuarios y/o grupos, y modificar sus permisos para ciertos archivos, siempre y cuando se cuente con los privilegios de administrador esperados al ejecutar el script.

Una de las principales diferencias encontradas para los comandos “chown” y “chmod” existentes con el mismo nombre en ambos tipos de sistema, es que en Linux tienen la opción adicional “-c” que macOS no tiene, que en caso de terminar sin errores retorna un valor distinto si encontró o no cambios en la asignación de permisos, es decir, si la nueva modificación solicitada afecta o no los permisos ya existentes, útil para mostrar con más detalle lo que realmente sucede.

Las Figuras 13 a 17 muestran los resultados de ingresar argumentos no válidos para la ejecución del script, mientras que las Figuras 18 a 20 muestran los resultados de validar la existencia tanto de usuarios como de grupos, así como de su creación en caso de no existir, o de la asignación de permisos respectiva en caso de existir.

Para la sección 3.3 acerca de servicios, únicamente se puede argumentar sobre las pruebas realizadas en el sistema GNU de Arch Linux, donde la Figura 31 muestra la forma de iniciar o finalizar el servicio solicitado, donde ambos scripts necesitan ejecución mediante “sudo” para contar con privilegios o credenciales de un administrador como el “root”.

La Figura 32 muestra un posible estado cuando el servicio es iniciado exitosamente, y crea el proceso de monitoreo que se ejecuta indefinidamente en el background, siempre y cuando el servicio no sea detenido.

La Figura 33 muestra una posible salida a un archivo CSV, resultado de crear el proceso de monitoreo mediante la inicialización del servicio, donde se guardan todos los eventos registrados para creación, modificación, y/o eliminación que ocurren dentro de un directorio especificado en el script. Para este caso, el directorio que se monitorea es “/tmp/”, y el archivo CSV se guarda en “/var/tmp/”.

La Figura 34 muestra otra forma de crear el proceso mediante la ejecución desde la terminal, que registra los eventos que ocurren en el directorio “/home/user/Desktop”, y que, en caso de solicitarlo en el script, se puede guardar en la ruta “/home/user/Downloads”, sin embargo, se muestran directo a pantalla en tiempo real.

5. CONCLUSIONES Y RECOMENDACIONES

Esta vez sí se recomienda ejecutar el script en otro sistema como macOS que también cuenta con Shell en Bash pero que no es GNU, con el fin de observar y poder comparar los resultados obtenidos. Sin embargo, únicamente se recomienda para el primer ejercicio de este laboratorio acerca de usuarios y permisos, ya que es el único script validado y probado para ambos tipos GNU vs non-GNU. En los otros ejercicios, no se incluyeron comandos alternativos para non-GNU, por lo que si se tratan de ejecutar en macOS se van a obtener errores inesperados de algunos comandos que no existen o sí existen, pero tienen diferencias importantes.

Para el tercer ejercicio acerca de servicios, no se recomienda guardar el archivo de log en la misma ruta monitoreada, ya que el comando parece no haber validado dicho caso, y en lugar de registrar únicamente el primer evento, sigue registrando la modificación del archivo indefinidamente y guardando líneas de eventos repetidos, lo cual provoca "overhead" y hace que el archivo se haga muy grande rápidamente, consumiendo exceso de recursos que se requieren libres para otros procesos.