Desafío 3:

Informe de modificaciones a un Script Bash que brinda información del sistema. Instrucciones de instalación y uso.

-Documentación-

Presentado por:

Andrés Burbano

Profesor:

Ezequiel González Rodríguez

Tutor:

Franco Congedo

Comisión:

73601

Bootcamp DevOps Engineer

EducaciónIT

2024

Informe de modificaciones a un Script Bash que brinda información del sistema. Instrucciones de instalación y uso.

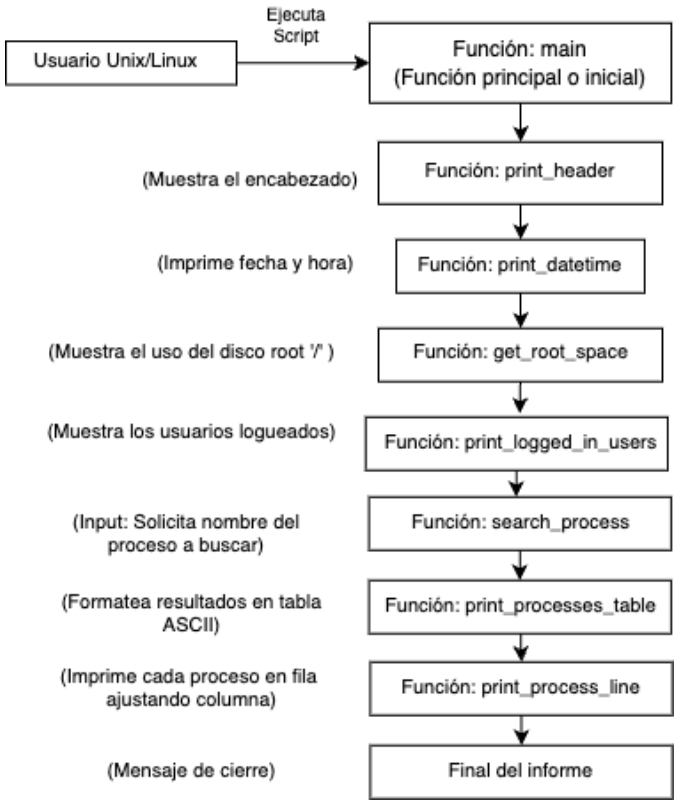
Tabla de contenido

1. Introducción

El presente documento muestra las modificaciones y optimizaciones implementadas a un script de Bash el cual genera un informe detallado del sistema, mostrando información como el uso de disco, los usuarios actualmente logueados y la búsqueda de procesos. La herramienta ha sido actualizada respecto a una versión previa, agregando manejo de excepciones, una salida formateada en tablas ASCII, y una estructura interna más limpia y mantenible.

El objetivo principal es proveer una herramienta robusta y fácil de interpretar que ayude a administradores de sistemas, ingenieros de soporte y otros profesionales de TI a obtener, de manera rápida, información relevante sobre el estado actual de un entorno Unix/Linux.

Diagrama de alto nivel que muestra el flujo del sistema:



Características Principales del script Bash

- **Reporte de sistema:** Muestra información básica del sistema como fecha, hora, uso de disco y usuarios conectados.

- **Búsqueda de procesos:** Permite al usuario ingresar el nombre de un proceso a buscar e imprime los resultados en una tabla formateada.

- **Formato Tabular ASCII:** Los resultados se presentan en tablas con ancho de columnas fijo y soporte de ajuste de texto en la columna de comandos, evitando salidas desalineadas.

- **Robustez y Manejo de Errores:** Uso de `set -euo pipefail`, manejo de señales (Ctrl+C), y manejo controlado de casos en los que no se encuentran procesos.

- **Modularidad:** Separación del código en funciones, con roles claros y mantenibles.

1. Explicación detallada de cambios y optimizaciones del script

A continuación se detallan las modificaciones realizadas respecto a la versión original, abarcando la reestructuración interna, el manejo de errores y la presentación de resultados.

1.1 Reestructuración en Funciones

- **Antes:** El script contenía bloques de código lineales, algunos comentarios, pero sin una clara separación lógica.

```
set -euo pipefail

print_header() {
    # imprimo un encabezado con el nombre de host.
    echo "================================"
    echo "Informe del sistema: $(hostname -s)"
    echo "================================"
} ← #5-10 print_header()

# Imprimir la fecha con el formato DD/MM/YYYY HH:MM:SS

echo "Fecha y Hora: $(date '+%d/%m/%Y %H:%M:%S')"
echo ""
get_root_space() {
    local point=$1
    # imprimo el espacio disponible de la unidad root (/)
    echo "Uso del Disco $(df -h $point | awk 'NR > 1 {print $7}')"
} ← #16-20 get_root_space()

# usuario logueados

echo "Estos son los usuario que estan usando el sistema"

who | awk '{print $1}'

# el comando free -m nos muestra la memoria (no esta disponible en mi mac)

while true; do
    read -p "Ingrese el proceso que desea buscar: " proceso
    if [[ -z "$proceso" ]]; then
        echo "Debe ingresar un nombre de proceso"
    else
        echo "Los datos del procesos son: "
        procesos_encontrados=$(ps aux | grep -i --color=auto "$proceso" | grep -v grep || true)

        if [[ -z "$procesos_encontrados" ]]; then # la evaluacion no esta siendo tomada.
            echo "No se encontraron procesos con el nombre '$proceso'. Vuelve a intentar"
        else
            echo "Se encontro el proceso estos son los resultados"
            echo "$procesos_encontrados"
            break
        fi
        echo $procesos_encontrados
    fi
done

main(){
    print_header
    get_root_space "/"
    echo "Fin del informe"
} ← #49-53 main()
main
```

- **Después:** Se crearon funciones con nombres descriptivos y roles bien definidos:
    - **print_header():** Imprime el encabezado del informe:

```bash
# Función para imprimir un encabezado del informe con el nombre del host
print_header() {
    echo "═══════════════════════════════════════════════════════"
    echo "              INFORME DEL SISTEMA: $(hostname -s)"
    echo "═══════════════════════════════════════════════════════"
} ← #27-31 print_header()
```

    - **print_datetime():** Muestra la fecha y hora actual:

```bash
# Función para imprimir la fecha y hora actual en formato DD/MM/YYYY HH:MM:SS
print_datetime() {
    echo "Fecha y Hora: $(date '+%d/%m/%Y %H:%M:%S')"
    echo ""
}
```

    - **get_root_space(mount_point):** Informa el uso del disco en la partición especificada.

```bash
# Imprimir el uso de disco de la partición raíz
get_root_space() {
    local mount_point="$1"
    local disk_usage
    # Se obtiene la columna correspondiente al uso (porcentaje)
    disk_usage=$(df -h "$mount_point" | awk 'NR > 1 {print $5}')

    echo "+-------------------------------------------------------+"
    echo "| Uso del Disco en $mount_point: $disk_usage"
    echo "+-------------------------------------------------------+"
    echo ""
} ← #40-50 get_root_space()
```

    - **print_logged_in_users():** Lista los usuarios actualmente conectados.

```bash
# Imprimir los usuarios actualmente logueados en el sistema
print_logged_in_users() {
    echo "+-------------------------------------------------------+"
    echo "| Usuarios actualmente logueados:"
    echo "+-------------------------------------------------------+"
    # Usar who y awk para listar los usuarios
    who | awk '{print "| " $1}'
    echo "+-------------------------------------------------------+"
    echo ""
} ← #53-61 print_logged_in_users()
```

- **search_process():** Solicita un proceso al usuario y muestra los resultados en tabla.

```
################################################################################
# Función para buscar proceso
################################################################################
#
# Esta función solicita al usuario el nombre de un proceso a buscar y muestra
# los resultados en una tabla formateada. Si no se encuentran resultados,
# se le pide al usuario que ingrese otro nombre hasta que se obtenga uno válido.

search_process() {
    while true; do
        read -r -p "Ingrese el proceso que desea buscar: " proceso
        echo ""

        # Validar que el usuario haya ingresado un nombre de proceso
        if [[ -z "$proceso" ]]; then
            echo "Debe ingresar un nombre de proceso. Inténtelo de nuevo."
            echo ""
            continue
        fi

        # Buscar el proceso ignorando mayúsculas/minúsculas. Excluimos la propia búsqueda (grep).
        # Usamos || true para que el script no finalice si grep no encuentra nada.
        procesos_encontrados=$(ps aux | grep -i "$proceso" | grep -v grep || true)

        if [[ -z "$procesos_encontrados" ]]; then
            echo "No se encontraron procesos con el nombre '$proceso'. Inténtelo de nuevo."
        else
            echo "Se encontraron los siguientes procesos con el nombre '$proceso':"
            print_processes_table "$procesos_encontrados"
            break
        fi
    done
} ← #208-232 search_process()
```

- **Funciones de apoyo (print_process_line(), print_processes_table(), print_header_table(), etc.)** para formatear y presentar los datos.

```
# Función para imprimir una fila de la tabla.
# Si la columna COMMAND excede el ancho, se partirá en varias líneas.
print_process_line() {
    local user="$1"
    local pid="$2"
    local cpu="$3"
    local mem="$4"
    local vsz="$5"
    local rss="$6"
    local tty="$7"
    local stat="$8"
    local start="$9"
    local time="${10}"
    shift 10
    local command="$*"

    # Usar fold para dividir la columna COMMAND en líneas de máximo COL_CMD_WIDTH caracteres
    # fold -s corta por palabras, evitando partir las palabras a la mitad si es posible.
    local folded_command
    folded_command=$(echo "$command" | fold -s -w "$COL_CMD_WIDTH")

    # El comando ahora puede tener varias líneas. Las procesaremos una por una.
    local first_line=true
    while IFS= read -r line; do
        if $first_line; then
            # Primera línea: imprimir todas las columnas
            printf "|%-${COL_USER_WIDTH}.${COL_USER_WIDTH}s|%-${COL_PID_WIDTH}.${COL_PID_WIDTH}s|%-${COL_CPU_WIDTH}.
                "$user" "$pid" "$cpu" "$mem" "$vsz" "$rss" "$tty" "$stat" "$start" "$time" "$line"
            first_line=false
        else
            # Siguientes líneas: imprimir las columnas vacías (para alineación) y solo COMMAND
            printf "|%-${COL_USER_WIDTH}s|%-${COL_PID_WIDTH}s|%-${COL_CPU_WIDTH}s|%-${COL_MEM_WIDTH}s|%-${COL_VSZ_WI
                "" "" "" "" "" "" "" "" "" "$line"
        fi
    done <<< "$folded_command"
} ← #123-156 print_process_line()
```

Estos cambios facilitan el mantenimiento, la lectura del código y el escalamiento de funcionalidades

1.2 Manejo de Errores y Excepciones

- **set -euo pipefail**: Asegura que el script se detenga en caso de errores inesperados, uso de variables indefinidas o fallos en pipes.

```
# set -euo pipefail:
# -e: Si algún comando retorna un estado distinto a 0, se detiene el script.
# -u: Si se usa una variable no definida, se considera error y se detiene el script.
# -o pipefail: Si un comando en una cadena (pipe) falla, se considera error general.
#
# Estas configuraciones ayudan a asegurar que cualquier fallo se detecte
# inmediatamente, evitando comportamientos inesperados.
#
set -euo pipefail
```

- **Manejo de interrupciones (trap)**: Al presionar `Ctrl+C` o al recibir una señal de terminación, el script imprime un mensaje claro y sale de forma ordenada.

```
# Atrapar señales de interrupción (ej. Ctrl+C) y terminación
trap 'echo " Script interrumpido por el usuario"; exit 1' INT TERM
```

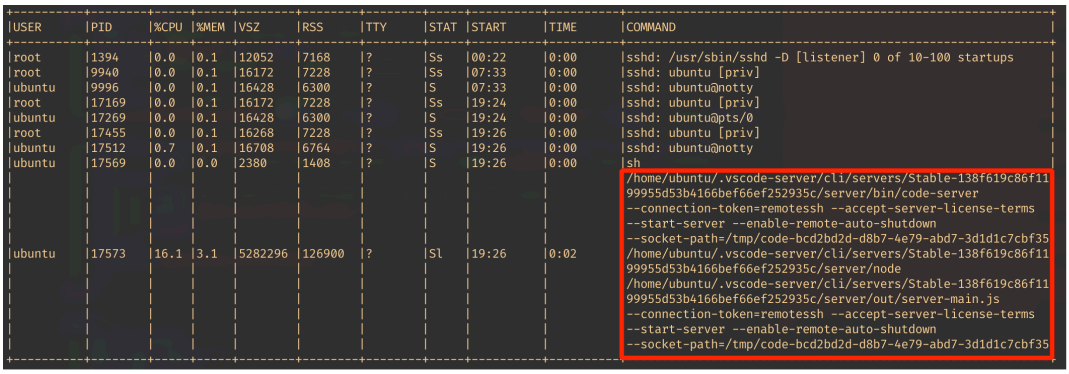
- **Validación de entrada en la búsqueda de procesos**: Si el usuario no ingresa un nombre de proceso o no se encuentran resultados, el script informa la situación y permite reintentos en lugar de fallar.

```
read -r -p "Ingrese el proceso que desea buscar: " proceso
echo ""

# Validar que el usuario haya ingresado un nombre de proceso
if [[ -z "$proceso" ]]; then
    echo "Debe ingresar un nombre de proceso. Inténtelo de nuevo."
    echo ""
    continue
fi
```

- **Ajustes en los comandos grep**: Se usa `|| true` para que la ausencia de procesos no detenga todo el script, sino que se maneje como un caso previsto.

```
# Buscar el proceso ignorando mayúsculas/minúsculas. Excluimos la propia búsqueda (grep).
# Usamos || true para que el script no finalice si grep no encuentra nada.
procesos_encontrados=$(ps aux | grep -i "$proceso" | grep -v grep || true)

if [[ -z "$procesos_encontrados" ]]; then
    echo "No se encontraron procesos con el nombre '$proceso'. Inténtelo de nuevo."
else
    echo "Se encontraron los siguientes procesos con el nombre '$proceso':"
    print_processes_table "$procesos_encontrados"
    break
fi
```

1.3 Mejoras en la Presentación de Datos



```
+----------+-------+-----+-----+--------+--------+--------+-----+---------+---------+-------------------------------------------------------------------------------------------------+
|USER      |PID    |%CPU |%MEM |VSZ     |RSS     |TTY     |STAT |START    |TIME     |COMMAND                                                                                          |
+----------+-------+-----+-----+--------+--------+--------+-----+---------+---------+-------------------------------------------------------------------------------------------------+
|root      |1394   |0.0  |0.1  |12052   |7168    |?       |Ss   |00:22    |0:00     |sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups                                          |
|root      |9940   |0.0  |0.1  |16172   |7228    |?       |Ss   |07:33    |0:00     |sshd: ubuntu [priv]                                                                              |
|ubuntu    |9996   |0.0  |0.1  |16428   |6300    |?       |S    |07:33    |0:00     |sshd: ubuntu@notty                                                                               |
|root      |17169  |0.0  |0.1  |16172   |7228    |?       |Ss   |19:24    |0:00     |sshd: ubuntu [priv]                                                                              |
|ubuntu    |17269  |0.0  |0.1  |16428   |6300    |?       |S    |19:24    |0:00     |sshd: ubuntu@pts/0                                                                               |
|root      |17455  |0.0  |0.1  |16268   |7228    |?       |Ss   |19:26    |0:00     |sshd: ubuntu [priv]                                                                              |
|ubuntu    |17512  |0.7  |0.1  |16708   |6764    |?       |S    |19:26    |0:00     |sshd: ubuntu@notty                                                                               |
|ubuntu    |17569  |0.0  |0.0  |2380    |1408    |?       |S    |19:26    |0:00     |sh                                                                                               |
|          |       |     |     |        |        |        |     |         |         |/home/ubuntu/.vscode-server/cli/servers/Stable-138f619c86f11                                     |
|          |       |     |     |        |        |        |     |         |         |99955d53b4166bef66ef252935c/server/bin/code-server                                               |
|          |       |     |     |        |        |        |     |         |         |--connection-token=remotessh --accept-server-license-terms                                       |
|          |       |     |     |        |        |        |     |         |         |--start-server --enable-remote-auto-shutdown                                                     |
|          |       |     |     |        |        |        |     |         |         |--socket-path=/tmp/code-bcd2bd2d-d8b7-4e79-abd7-3d1d1c7cbf35                                     |
|ubuntu    |17573  |16.1 |3.1  |5282296 |126900  |?       |Sl   |19:26    |0:02     |/home/ubuntu/.vscode-server/cli/servers/Stable-138f619c86f11                                     |
|          |       |     |     |        |        |        |     |         |         |99955d53b4166bef66ef252935c/server/node                                                          |
|          |       |     |     |        |        |        |     |         |         |/home/ubuntu/.vscode-server/cli/servers/Stable-138f619c86f11                                     |
|          |       |     |     |        |        |        |     |         |         |99955d53b4166bef66ef252935c/server/out/server-main.js                                            |
|          |       |     |     |        |        |        |     |         |         |--connection-token=remotessh --accept-server-license-terms                                       |
|          |       |     |     |        |        |        |     |         |         |--start-server --enable-remote-auto-shutdown                                                     |
|          |       |     |     |        |        |        |     |         |         |--socket-path=/tmp/code-bcd2bd2d-d8b7-4e79-abd7-3d1d1c7cbf35                                     |
+----------+-------+-----+-----+--------+--------+--------+-----+---------+---------+-------------------------------------------------------------------------------------------------+
```

Si la ubicación del Command excede el tamaño de la columna se divide en varios renglones

Columnas de ancho fijo en la tabla

- **Tablas ASCII**: Los procesos se muestran ahora en una tabla con columnas de ancho fijo, encabezados claros y bordes ASCII.

- **Ajuste de texto (wrap)**: La columna COMMAND, que suele ser la más larga, se ajusta a múltiples líneas si excede el ancho definido, sin desformar la tabla.

- **Columnas con Espacios Fijos**: Esto evita salidas desalineadas y hace que la lectura sea más sencilla, mejorando la presentación visual.

1.4 Comentarios y Claridad del Código

- **Comentarios detallados**: Se han agregado explicaciones claras en cada sección del código, no solo indicando “qué” hace cada parte, sino también “para qué” se ha implementado de esa forma.

```
################################################################################
# Funciones para formatear la tabla de procesos
################################################################################
#
# Se define un ancho fijo para cada columna, y se imprime una tabla ASCII.
#
# Columnas del comando ps aux:
# USER       PID  %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
#
# Definimos anchos de columna aproximados:
# USER: 10
# PID:  8
# %CPU: 6
# %MEM: 6
# VSZ:  8
# RSS:  8
# TTY:  8
# STAT: 8
# START:10
# TIME: 10
# COMMAND: 50 (con ajuste de línea si excede)
#
# Nota: El COMMAND puede contener espacios. Por ello, se capturan todos los
# campos desde el 11 en adelante.
```

- **Separación Lógica**: La división del código en secciones con encabezados y comentarios contribuye a una mejor comprensión del flujo general.

```
# Función para repetir un carácter n veces
repeat_char() {…
} ← #102-106 repeat_char()

# Función para imprimir la línea divisoria de la tabla
print_divider() {…
}

# Función para imprimir el encabezado de la tabla
print_header_table() {…
} ← #114-119 print_header_table()

# Función para imprimir una fila de la tabla.
# Si la columna COMMAND excede el ancho, se partirá en varias líneas.
print_process_line() {…
} ← #123-156 print_process_line()

# Función para imprimir todos los procesos encontrados en una tabla formateada
print_processes_table() {…
} ← #159-198 print_processes_table()

################################################################################
# Función para buscar proceso
################################################################################
#
# Esta función solicita al usuario el nombre de un proceso a buscar y muestra
# los resultados en una tabla formateada. Si no se encuentran resultados,
# se le pide al usuario que ingrese otro nombre hasta que se obtenga uno válido.

search_process() {…
} ← #208-232 search_process()

################################################################################
# Función principal
################################################################################

main() {
    print_header
    print_datetime
    get_root_space "/"
    print_logged_in_users
    search_process
    echo "Fin del informe."
} ← #238-245 main()

################################################################################
# Ejecución del script
################################################################################

main
```

2. Instalación

2.1 Requisitos Previos

Antes de comenzar, se debe asegurar que se cumplen los siguientes requisitos:

2.1.1 Sistema Operativo compatible:

Se debe contar con un SO compatible con Bash: Unix/Linux (Ej: Ubuntu, Debian, CentOS, macOS)

2.1.2 Shell:

Bash en su versión 4 o superior

2.1.3 Herramientas estándar:

Tener instalado en el SO las herramientas estándar: **ps, df, awk, fold, grep, date, who** (Todos estos comandos suelen venir preinstalados en la mayoría de sistemas tipo Unix)

2.2 Pasos Instalación

2.2.1 Clonar o descargar el script de Bash

Se debe descargar o clonar este script desde el siguiente enlace en GitHub: https://raw.githubusercontent.com/andres-b-devops/desafio-03/refs/heads/main/informe-sistema.sh

2.2.2 Dar permisos de ejecución

```
chmod +x informe-sistema.sh
```

2.2.3 Ubicar el script en el PATH (opcional)

Puede colocar el script en una ubicación dentro de su $PATH, como: `/usr/local/bin/`.

Ej: sudo cp informe_sistema.sh /usr/local/bin/informe_sistema

De esa forma se puede ejecutar simplemente con: `informe_sistema`

3. Uso del script

3.1 Ejecución:

En la terminal escribir lo siguiente: `./informe_sistema.sh`

O si se ha instalado en el PATH, simplemente: `informe_sistema`

Una vez ejecutado, el script mostrará un encabezado con el nombre del servidor, la fecha y hora actual (con el formato día/mes/año hora:minutos:segundos), también el uso del disco en la partición raíz y los usuarios conectados. Luego pedirá que el usuario ingrese el nombre de un proceso a buscar, si no encuentra coincidencias, el script entrará en un ciclo donde pedirá volver a reintentar hasta encontrar un resultado válido.

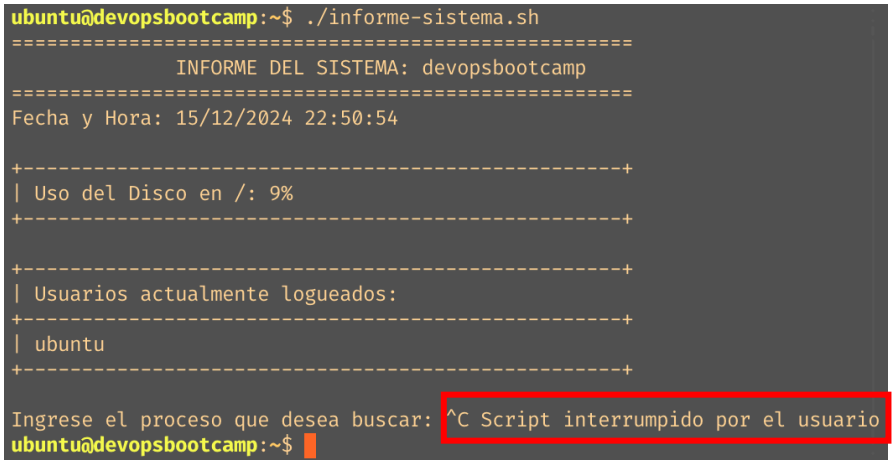
3.2 Resultado de ejecución:

A continuación, se presentan ejemplos de uso en una instancia multipass llamada `devopsbootcamp`:

3.2.1 **Ejemplo 1:** buscando el proceso: `ssh` mostrará la información separada por columnas, agregando los encabezados.

```
ubuntu@devopsbootcamp:~$ ./informe-sistema.sh
==================================================
              INFORME DEL SISTEMA: devopsbootcamp
==================================================
Fecha y Hora: 15/12/2024 22:43:38

+--------------------------------------------------+
| Uso del Disco en /: 9%
+--------------------------------------------------+

+--------------------------------------------------+
| Usuarios actualmente logueados:
+--------------------------------------------------+
| ubuntu
+--------------------------------------------------+

Ingrese el proceso que desea buscar: ssh

Se encontraron los siguientes procesos con el nombre 'ssh':
+----------+--------+-----+-----+--------+--------+--------+-----+----------+----------+----------------------
|USER      |PID     |%CPU |%MEM |VSZ     |RSS     |TTY     |STAT |START     |TIME      |COMMAND
+----------+--------+-----+-----+--------+--------+--------+-----+----------+----------+----------------------
|root      |866     |0.0  |0.1  |16168   |7100    |?       |Ss   |15:55     |0:00      |sshd: ubuntu [priv]
|ubuntu    |1077    |0.0  |0.1  |16424   |6300    |?       |S    |15:55     |0:00      |sshd: ubuntu@notty
|root      |1394    |0.0  |0.1  |12052   |7168    |?       |Ss   |16:03     |0:00      |sshd: /usr/sbin/sshd
|root      |3040    |0.0  |0.1  |16172   |7228    |?       |Ss   |19:02     |0:00      |sshd: ubuntu [priv]
|ubuntu    |3096    |0.0  |0.1  |16428   |6304    |?       |S    |19:02     |0:00      |sshd: ubuntu@pts/2
+----------+--------+-----+-----+--------+--------+--------+-----+----------+----------+----------------------

Fin del informe.
ubuntu@devopsbootcamp:~$
```

3.2.2 **Ejemplo 2:** Cancelando el proceso con la combinación de teclas: ctrl+c, muestra mensaje indicando que se ha interrumpido.



3.2.3 **Ejemplo 3:** Ingresando varios procesos que no existen (‘saludo’, ’despedida’, ’hola’, chao’) para verificar que vuelva a solicitarle al usuario el ingreso de este.