SISTEMAS OPERATIVOS SEGUNDA ENTREGA.

Configuracion SSH.

A continuación se proporciona una guía detallada sobre el uso de un playbook de Ansible diseñado para configurar un servidor SSH, instalar el cliente de MySQL, ajustar la conexión a la base de datos y agregar una clave SSH para un usuario específico en un servidor CentOS 7. Este playbook es una herramienta poderosa para la administración de servidores, ya que automatiza tareas comunes de configuración y permite una gestión eficiente y coherente de servidores.

Funcionalidades del Playbook.

Configuración de Puerto SSH:

Permite establecer un puerto personalizado para la conexión SSH. Esta característica es útil cuando se necesita cambiar el puerto predeterminado para mejorar la seguridad del servidor.

Instalación del Cliente de MySQL:

Nuestro playbook de Ansible simplifica el proceso de instalación del cliente de MySQL en el servidor Centos7. Esta instalación es esencial para permitir la interacción efectiva con bases de datos MySQL desde el servidor. Haciendo innecesario realizar este proceso manualmente, lo que ahorra tiempo y garantiza que el cliente de MySQL esté disponible y listo para su uso.

Gestión de Claves SSH:

Proporciona la capacidad de agregar una clave SSH a un usuario específico. Esto simplifica el proceso de autenticación y permite el acceso seguro al servidor.

Ajustes de conexión a la Base de Datos:

Permite configurar y ajustar la conexión a una base de datos MySQL. Esto incluye la creación de usuarios, contraseñas y permisos necesarios para acceder a la base de datos.

```
  hosts: servidores
  become: yes
  tasks:
    - name: instalar claves en el sistema
      authorized_key:
              user: root
              state: present
              key: "{{ lookup('file', '/root/.ssh/id_rsa.pub') }}" #<-la clave id_rsa para el usuario "
root" debe de ser generada desde ssh usando ssh-keygen -t rsa
    - name: Configuracion puerto SSH
      become_user: root
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: '#?Port'
        line: 'Port 22'
      notify: Reinicio de SSH

    - name: Instalacion de cliente MYSQL
      yum:
        name: mysql
        state: present

    - name: Conexion del cliente a la base de datos
      mysql_user:
        name: root
        password: 12345
        host: localhost
        state: present

    - name: Reinicio de servicio SSH
      become: yes
      service:
        name: sshd
        state: restarted
~
~
                                                                              35,1          Final
```

Generacion de clave o “Key”

Si queremos autorizar al usuario que escogimos (en este caso el usuario llamado “root”) al momento de realizar nuestro playbook al servicio ssh es necesario generar una clave para dicho usuario.

Esto quiere decir que debemos de hacer login dentro de ssh con un usuario del tipo administrador y proceder a escribir el siguiente comando:

ssh-keygen -t rsa: Este comando genera la clave o “key” la que luego se almacenara de un archivo el cual nosotros escogamos la ruta. En el siguiente ejemplo se muestra la ruta /home/anees/.ssh/id_rsa dicha ruta deberá de ser colocada dentro de la línea **key:** de nuestro playbook. De esa manera autorizamos al usuario escogido anterior mente en la línea **user:** a acceder al servicio ssh.

```
Connection to 192.168.1.192 closed.
[anees@localhost ansible]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/anees/.ssh/id_rsa):
```

Medios de respaldo de datos y disponibilidad.

En la gestión de servidores CentOS7, es esencial contar con una estrategia sólida para respaldar datos a largo plazo y garantizar una alta disponibilidad, se explorarán las estrategias fundamentales en esta área, centrándonos en cinco aspectos clave.

Almacenamiento Redundante:

La implementación de un sistema de almacenamiento RAID es fundamental. RAID 1, conocido como espejo, ofrece redundancia al duplicar los datos en dos discos duros o más. Por otro lado, RAID 6 utiliza paridad distribuida para brindar tolerancia a fallos incluso en la falla de dos discos. Estas configuraciones protegen contra pérdidas de datos y garantizan la disponibilidad continua.

Copias de Seguridad Regulares:

La realización de copias de seguridad regulares es esencial para la protección de datos. Herramientas como rsync y tar son útiles para este propósito. Además, almacenar copias de seguridad en ubicaciones geográficamente separadas reduce el riesgo de pérdida de datos debido a desastres naturales.

Virtualización y Migración:

La virtualización de servidores permite la fácil migración de sistemas y datos en caso de fallo del servidor físico. Esto proporciona una alta disponibilidad al minimizar el tiempo de inactividad en situaciones de emergencia. La virtualización puede lograrse mediante soluciones como VMware o VirtualBox.

Encriptación de Datos:

La encriptación de datos es crucial tanto en reposo como en tránsito. Garantizar que los datos estén protegidos contra accesos no autorizados es esencial para la seguridad de los datos. El uso de protocolos seguros como HTTPS para la transmisión de datos y la implementación de sistemas de archivos encriptados son prácticas recomendadas.

Actualizaciones y Parches:

El mantenimiento constante del sistema es fundamental para mitigar vulnerabilidades y garantizar un rendimiento óptimo. Se deben aplicar regularmente actualizaciones de seguridad y parches para mantener el servidor seguro y en funcionamiento sin problemas.

Conclusión:

La combinación de almacenamiento redundante, copias de seguridad regulares, virtualización y migración, encriptación de datos y mantenimiento del sistema en servidores brinda una estrategia para respaldar datos a largo plazo y garantizar una alta disponibilidad. Estas prácticas son esenciales para proteger la accesibilidad de los datos. La implementación adecuada de estas estrategias proporciona la confianza necesaria para enfrentar los desafíos que puedan surgir en la gestión de servidores.

Rutinas de Backups.

A continuación se procederá de forma detallada la puesta en uso del script el cual llamamos "Backups_rutina.sh" de modo que dicho script sea ejecutado automáticamente por nuestro sistemas operativo centos7 sin la necesidad de la presencia de un operador en el momento, para esto se deberá de contar con el el uso del programa "cron" el cual nos proporcionará dicha automatización.

Deberemos de crear archivo utilizando el comando touch nombrando dicho archivo con su extensión .sh, Ingresamos al archivo utilizando el gestor de textos vi. Una vez dentro comenzamos a incluir los siguientes comandos:

#!/bin/bash: Nos permitirá que dicho archivo sea interpretado como un script por bash.

Hora=$(date + '%H:%M%:S'): Nos permite crear una variable llamada "hora" la cual nos dira la hora exacta.

"$Hora": Variable a utilizar

Echo -e: Comando en la terminal de centos7 que se utiliza para imprimir texto en pantalla.

Playbooks_dir="$HOME/etc/ansible/hosts":

Playbooks_dir: es una variable que almacena una ruta de directorio.

$HOME es una variable de entorno que generalmente representa el directorio de inicio del usuario.

La ruta completa etc/ansible/hosts se agrega al directorio de inicio usando $HOME. (**IMPORTANTE:** utilizaremos la ruta etc/ansible/hosts ya que es la ruta creada por defectos en ansible en la cual se almacenan dichos playbooks)

Backup_dir="$HOME/ruta_del_backup/backups/Carpeta_de_playbooks":

Backup_dir es otra variable que almacena una ruta de directorio.

Mkdir -p "$backup_dir": En caso de no existir todavía un directorio el cual alamcene nuestros playbooks, este comando creara uno.

Comando Adicional:

Zip -r “$playbooks_dir” “$backup_dir”: Permite comprimir nuestro backup y lo envia a el directorio el cual queremos se almacene usando las variables anteriormente creadas.

Zip: comando que se utiliza para crear archivos ZIP. (Es importante asegurarse de que el comando zip esté instalado en tu sistema antes de usarlo).

-r: Esta opción indica a zip que debe comprimir recursivamente todos los archivos y subdirectorios dentro del directorio que deseas comprimir.

Representación grafica del contenido del script.

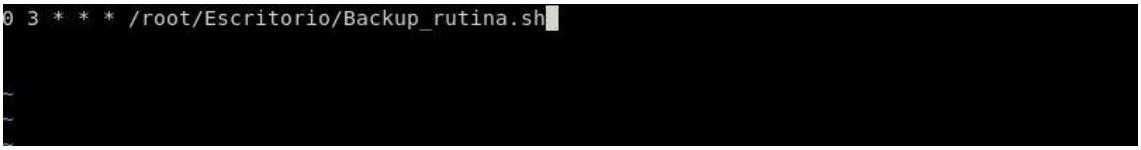
```
#!/bin/bash

#Script de Rutina ligada a backup
#Zennet
#Version 1.0

GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'

hora=$(date +'%H:%M:%S')

echo -e "${YELLOW} ------  Bienvenido al servicio de rutinas diarias. ----- ${NC}"
echo -e " "
echo -e  "${GREEN} Se procede a la ejeccucion de respaldo de archivos sensibles... ${NC}"

playbooks_dir="$HOME/etc/ansible/hosts"

backup_dir="$HOME/Escritorio/backups/ansible_playbooks"

echo -e "${YELLOW} ------  Hora de realizado el respaldo ----- ${NC}"
echo -e "${YELLOW} "$hora" ${NC}"
echo -e " "
echo -e " "
echo -e "${GREEN} Creando directorio para backups... ${NC}"

mkdir -p "$backup_dir"

echo -e "${GREEN} Comprimiendo... ${NC}"
echo -e " "
zip -r "$playbooks_dir" "$backup_dir"

echo -e "${GREEN} Backup rutinario ejecutado correctamente ${NC}"
echo -e " "
echo -e "${GREEN} Archivos de respaldo almacenados en: "$backup_dir" ${NC}"
```

```
------  Bienvenido al servicio de rutinas diarias. -----

 Se procede a la ejeccucion de respaldo de archivos sensibles...
 ------  Hora de realizado el respaldo -----
 12:42:00


 Creando directorio para backups...
 Comprimiendo...

updating: root/Escritorio/ansible_playbooks/ (stored 0%)
updating: root/Escritorio/ansible_playbooks/ServicioSSH (stored 0%)
  adding: root/Escritorio/ansible_playbooks/ADHD/ (stored 0%)
  adding: root/Escritorio/ansible_playbooks/Ansible_computos.sh (stored 0%)
  adding: root/Escritorio/ansible_playbooks/SUSELinux/ (stored 0%)
 Backup rutinario ejecutado correctamente

 Archivos de respaldo almacenados en: /root/Escritorio/backups/ansible_playbooks
```

Creación de rutina utilizando cron:

1) Ejecutamos el programa cron utilizando el comando: crontab -e

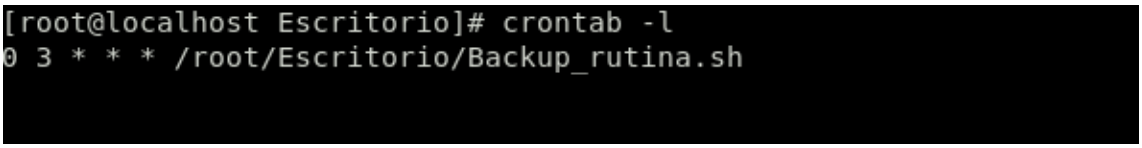
2) automáticamente nos dirigirá a un archivo con el editor de vi tal como se muestra en la siguiente imagen:



3) Nuestro siguiente paso para realizar la rutina será el seteo de el horario el cual querramos definir para el inicio automático de nuestro script. Para este ejemplo nuesto script se ejecutará diariamente a la hora: 03:00am. (Para realizar el calculo necesario puede consultarse de manera gratuita por medio del siguiente link: AQUI) Luego se procede a definir la ruta de nuestro archivo en nuestro caso, /root/Escritorio/Backup_rutina.sh.

4) Al finalizar nuestra tareas usaremos el comando WQ! Que nos permitirá guardar y salir de crontab, una vez hecho nuestra tarea quedará programada con éxito.

5) En caso de querer verificar si nuestra/s rutinas se encuentran operativas utilizaremos el comando: crontab -L el cual desplegara la lista de tareas en ejecución.



Backup de BDD.

A continuación se muestra el playbook creado en el gestor de textos vi que nos permitirá el realizar exitosamente el backup o respaldo de una base de datos, para este ejemplo tomamos la base de datos llamada “MariaDB” utilizando el siguiente comando:

mysqldump: Este es el comando que se utiliza para ejecutar la utilidad de copia de seguridad de MySQL.

-u root: Esto especifica el usuario de MySQL que se utilizará para realizar la copia de seguridad. En este caso, se está utilizando el usuario “root”.

-**p:** Esta opción indica a MySQL que debe solicitar una contraseña. Después de ingresar el comando, el sistema te pedirá que ingreses la contraseña la cual en este caso fue “12345”

--database MariaDB: Esta parte del comando especifica la base de datos que se va a respaldar. En este caso, se está respaldando una base de datos llamada “MariaDB”, (Debes reemplazar “MariaDB” con el nombre de la base de datos que deseas respaldar).

>: Operador lógico que redirige la salida del comando a un archivo.

/root/MariaDBBackup.sql: Esta es la ruta y el nombre del archivo de respaldo donde se guardarán los datos de la base de datos.

```
---
- name: Script realiza respaldo base de datos
  hosts: servidores
  gather_facts: no
  tasks:

    - name: Creacion de backup de la base de datos
      become: yes
      become_user: root   <--- Aqui se cambia por el nombre de usuario de Mysql
      shell: mysqldump -u root -p 12345 --database MariaDB > /root/MariaDBBackup.sql
~
~
~
~
~
~
~
~
~
~
~
```

Creacion de rutina: Al igual que en nuestro caso anterior se creará una tarea utilizando cron la cual permita ejectuar dicho playbook de forma automática ya sea en una fecha, hora o tiempo en especifico.

Al momento de crear nuestra nueva tarea de rutina no es necesario crear un nuevo archivo crontab ya que cron posee una arquitectura la cual puede ejecutar varias tareas sean del formato o tipo que sean.

Con esto en mente procedemos a utilizar el comando: crontab -e y redactar la automatización de nuestra tarea en una nueva línea tal como se muestra en la imagen:

```
0 3 * * * /root/Escritorio/Backup_rutina.sh
0 4 * * * /root/Backup_bdd.yaml
~
~
~
```

Para este ejemplo se dejo un margen de unos 60 minutos entre backups ya que se busca el ahorro de recursos de nuestro servidor. En caso de ser necesario podremos verificar el estado de nuestras tareas utilizand crontab -l.

```
[root@localhost ~]# crontab -l
0 3 * * * /root/Escritorio/Backup_rutina.sh
0 4 * * * /root/Backup_bdd.yaml
```

Script centro de computos.

Introducción.

El script en cuestión desempeña un papel fundamental al proporcionar información detallada sobre nuestro sistema. Esto tiene diversas aplicaciones útiles tales como:

Monitoreo del Sistema: Permite supervisar el estado general del sistema. Los administradores pueden usarlo para mantener un ojo en la salud del servidor.

Resolución de Problemas: En situaciones de diagnóstico y resolución de problemas, el script brinda detalles valiosos sobre la configuración del sistema y el uso de recursos, lo que ayuda a identificar posibles problemas.

Verificación de Recursos: Es útil para verificar cómo se están utilizando recursos críticos como la CPU, la memoria y el espacio en disco, lo que es esencial para garantizar que el servidor funcione de manera eficiente.

Documentación y Registro: Facilita la documentación y el registro de la configuración y el estado del sistema en un momento dado. Esto es beneficioso para el mantenimiento y la auditoría del sistema.

En resumen, este script es una herramienta versátil que proporciona información rápida y relevante, Esto es fundamental para la monitorización, el diagnóstico de problemas y la gestión efectiva de recursos.

Uso de comandos dentro del Script.

echo -e “Hostname: $(hostname)”

Muestra el nombre del host del sistema, seguido por el nombre del host obtenido mediante el comando hostname.

echo -e "Sistema Operativo: $(cat /etc/os-release | grep “PRETTY_NAME” | cut -d”” -f)”

Muestra el sistema operativo, seguido por el nombre del sistema operativo obtenido del archivo /etc/os-release.

echo -e “Kernel: $(uname -r)”

Muestra la versión del kernel, obtenida mediante el comando uname -r.

echo -e “Memoria Total: $(free -m | awk 'NR==2 {print $2 “ MB”}')”

Muestra la cantidad total de memoria, obtenida del comando free -m.

***echo -e “CPU: $(lscpu | grep “Model name” | cut -d':' -f 2 | sed 's/^ //')”**

Muestra información sobre la CPU, incluyendo el modelo de la CPU. La información se obtiene del comando lscpu.

echo -e "Memoria Usada: **$(free -m | grep “Mem:”** | awk '{print $3 “MB”}') / Memoria Total: **$(free -m | grep “Mem:” | awk '{print $2 “ MB”}')”**

Muestra la cantidad de memoria usada y la memoria total. Los valores se obtienen del comando free.

Who:

Ejecuta el comando who para mostrar información detallada sobre los usuarios conectados.

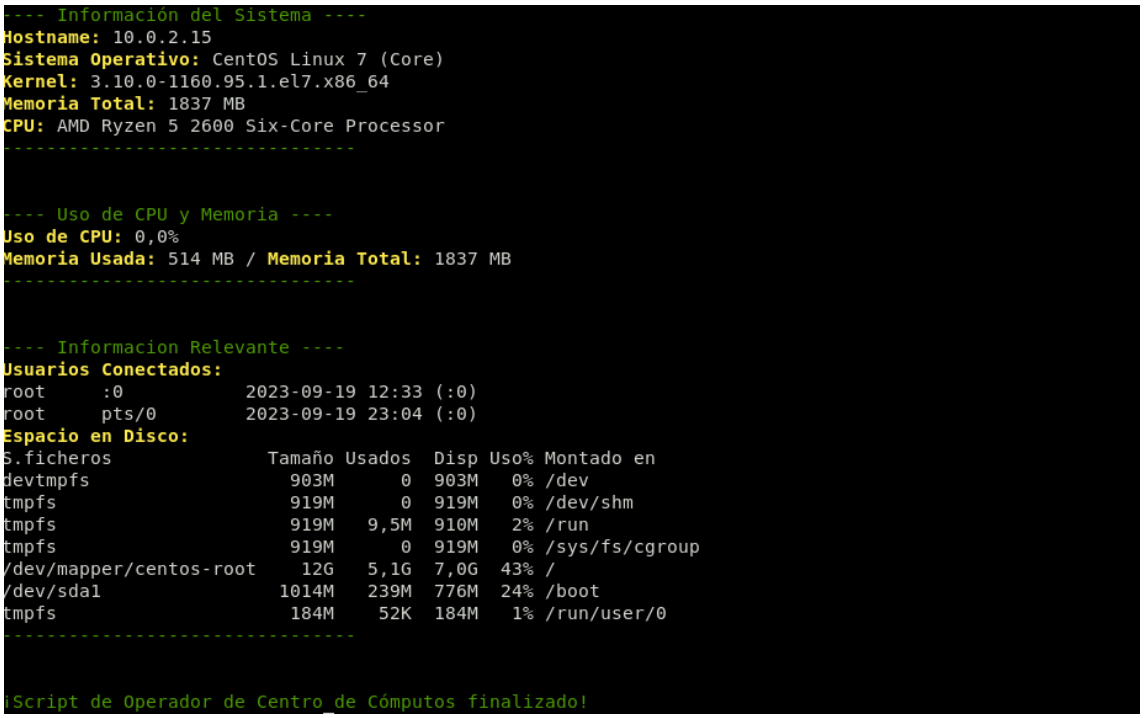
df -h:

El comando df -h muestra una lista de sistemas de archivos montados y su uso de espacio en disco.

Perspectiva grafica.

Imágenes del script en cuestión puesto en función obteniendo así la información del sistema.

```
#!/bin/bash

#Script de centro de computos
#Zennet
#Version 1.0

GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m'


echo -e "${GREEN}---- Información del Sistema ----${NC}"
echo -e "${YELLOW}Hostname:${NC} $(hostname)"
echo -e "${YELLOW}Sistema Operativo:${NC} $(cat /etc/os-release | grep "PRETTY_NAME" | cut -d'"' -f 2)"
echo -e "${YELLOW}Kernel:${NC} $(uname -r)"
echo -e "${YELLOW}Memoria Total:${NC} $(free -m | awk 'NR==2 {print $2 " MB"}')"
echo -e "${YELLOW}CPU:${NC} $(lscpu | grep "Model name" | cut -d':' -f 2 | sed 's/^ *//')"
echo -e "${GREEN}--------------------------------${NC}"
echo -e " "
echo -e " "
echo -e "${GREEN}---- Uso de CPU y Memoria ----${NC}"
echo -e "${YELLOW}Uso de CPU:${NC} $(top -b -n 1 | grep "Cpu(s)" | awk '{print $2 "%"}')"
echo -e "${YELLOW}Memoria Usada:${NC} $(free -m | grep "Mem:" | awk '{print $3 " MB"}') / ${YELLOW}Memo
ria Total:${NC} $(free -m | grep "Mem:" | awk '{print $2 " MB"}')"
echo -e "${GREEN}--------------------------------${NC}"
echo -e " "
echo -e " "
echo -e "${GREEN}---- Informacion Relevante ----${NC}"
echo -e "${YELLOW}Usuarios Conectados:${NC}"
who
echo -e "${YELLOW}Espacio en Disco:${NC}"
df -h
echo -e "${GREEN}--------------------------------${NC}"
echo -e " "
echo -e " "
echo -e "${GREEN}¡Script de Operador de Centro de Cómputos finalizado!${NC}"
```



Shell script modular.

Introduccion a Shell script.

Un shell script es un archivo de texto que contiene una secuencia de comandos que se ejecutan en una línea de comandos de un sistema operativo Unix o Unix-like, como Linux. La modularidad en un shell script se refiere a la práctica de dividir el script en módulos o funciones independientes, lo que facilita la organización y el mantenimiento del código.

Declaración de intérprete: El script comienza con la especificación del intérprete que se utilizará para ejecutar el código, comúnmente **#!/bin/bash** para indicar que se utilizará el shell Bash.

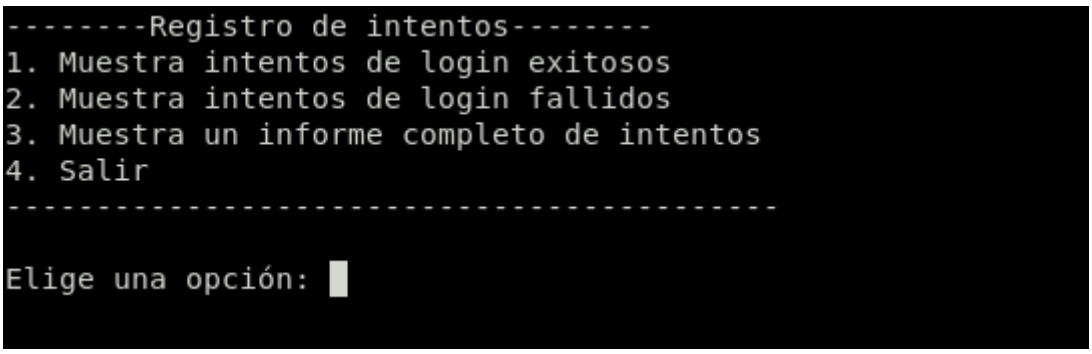
Módulos o funciones: El script se divide en módulos o funciones independientes que realizan tareas específicas. Cada función se define con el formato.

Formacion del Shell script: El siguiente script modular contendrá varios scripts de forma fragmentada/modular los cuales realizaran la tarea de recolectar los intentos de login a nuestra terminal o sistema. Dicho script se separa en cuatro modulos, dos son destinados a los intentos de login, uno destinado a nuestro menú y por ultimo tendremos nuestro script principal que se encarga de llamar a cada modulo individual.

Menu.

Opciones del Menú: El menú presenta cuatro opciones numeradas.

Solicitud de Elección: Después de mostrar las opciones, el script utiliza read -p para solicitar al usuario que elija una opción ingresando un número. La variable opcion almacenará la elección del usuario.



Principal.

Fuentes (Sources):

El script utiliza las instrucciones source para incluir otros archivos de script en su ejecución.

source menu.sh incluye un archivo llamado menu.sh, que contiene funciones relacionadas con la interfaz de usuario.

source login_exitoso.sh incluye un archivo llamado login_exitoso.sh, que contiene funciones para analizar registros de inicio de sesión exitosos.

source login_fallidos.sh incluye un archivo llamado login_fallidos.sh, que contiene funciones para analizar registros de inicio de sesión fallidos.

Bucle Principal:

El script contiene un bucle while que se ejecuta indefinidamente (while true) para mantener el programa en funcionamiento hasta que el usuario decida salir.En cada iteración del bucle, se llama a la función menu para mostrar un menú interactivo al usuario y se le pide que elija una opción.Un case se utiliza para manejar la opción seleccionada por el usuario.

Ejecución de Funciones:

Dependiendo de la opción elegida por el usuario, se llaman a funciones específicas. Por ejemplo, si el usuario elige la opción 1, se llama a la función login_exitoso para analizar registros de inicio de sesión exitosos. Lo mismo ocurre para la opción 2 y la función login_fallidos.

```
#!/bin/bash
RUTA="/var/log/secure*"
source menu.sh
source login_exitoso.sh
source login_fallidos.sh

while true; do
    menu
    case $opcion in
    1) login_exitoso.sh;;
    2) login_fallidos.sh ;;
    # ... Llama a las demás funciones según la opción ...
    3) exit ;;
    *) echo "Opción inválida" ;;
    esac
    mostrar_informe
done
```

Entradas exitosas.

La función login_exitoso se encarga de realizar el siguiente proceso.

Variables Internas:

Dentro de la función, se definen dos variables locales:

suser: Esta variable se utiliza para almacenar las líneas de registro que contienen la cadena "Accepted password". Estas líneas indican que un usuario ha tenido un inicio de sesión exitoso utilizando una contraseña.

scount: Esta variable se utiliza para almacenar todas las líneas de registro que contienen la cadena "Accepted". Esto incluye tanto los inicios de sesión exitosos con contraseña como otros métodos de autenticación exitosos.

Comandos grep:La función utiliza comandos grep para buscar cadenas específicas en el archivo de registro definido por la variable $RUTA. grep "Accepted password" $RUTA busca las líneas de registro que contienen la cadena "Accepted password" y las almacena en la variable suser. grep "Accepted" $RUTA busca todas las líneas de registro que contienen la cadena "Accepted" y las almacena en la variable scount.

```
#!/bin/bash
RUTA="/var/log/secure*"
login_exitoso() {
    suser=$(grep "Accepted password" $RUTA)
    scount=$(grep "Accepted" $RUTA)
}
```

Entradas fallidas.

La función login_fallidos se encarga de realizar el siguiente proceso:

Variables Internas:

Dentro de la función, se definen tres variables locales.

tuser: Esta variable se utiliza para almacenar las líneas de registro que contienen las cadenas "Accepted" o "Failed". Estas líneas indican intentos de inicio de sesión exitosos o fallidos.

fuser: Esta variable se utiliza para almacenar las líneas de registro que contienen la cadena "Failed password". Estas líneas indican intentos de inicio de sesión fallidos debido a contraseñas incorrectas.

fcount: Esta variable se utiliza para almacenar todas las líneas de registro que contienen la cadena "Failed". Esto incluye tanto los intentos de inicio de sesión fallidos con contraseñas incorrectas como otros tipos de fallas.

Comandos grep:

La función utiliza comandos grep para buscar cadenas específicas en el archivo de registro definido por la variable $RUTA.

grep "Accepted|Failed" $RUTA busca las líneas de registro que contienen las cadenas "Accepted" o "Failed" y las almacena en la variable tuser.

grep "Failed password" $RUTA busca las líneas de registro que contienen la cadena "Failed password" y las almacena en la variable fuser.

grep "Failed" $RUTA busca todas las líneas de registro que contienen la cadena "Failed" y las almacena en la variable fcount.

```
#!/bin/bash
RUTA="/var/log/secure*"
login_fallidos() {
    tuser=$(grep "Accepted|Failed" $RUTA)
    fuser=$(grep "Failed password" $RUTA)
    fcount=$(grep "Failed" $RUTA)
}
```