EL DIMONI

Carlos Cabrera Domingo

Versión: 1.1

Fecha: 7/2/24

Tabla de Contenidos

Introducción

En la presente práctica, exploramos la creación de un script que se desplegará como servicio. En otras palabras, construiremos un programa que se ejecutará automáticamente al encender la máquina virtual, realizando funciones específicas definidas en el script. En este caso particular, nos enfocaremos en:

Crear un programa capaz de recibir argumentos, entre ellos: **start | stop | restart | status.**

Manejar adecuadamente casos de error, como la ausencia de argumentos o la recepción de argumentos no esperados.

Implementar la funcionalidad 'start': Mostrar un mensaje si el programa ya está arrancado; de lo contrario, escribir en un archivo de registro el mensaje "Arrancando el programa..." y, cada 2 segundos, informar "Estoy vivo". Gestionar la funcionalidad 'stop': Mostrar un mensaje si el programa está detenido; en caso de estar arrancando, escribir en el registro el mensaje "Deteniendo programa" y detener la ejecución del programa. Realizar la funcionalidad 'restart': Detener y arrancar el programa. Implementar la funcionalidad 'status': Indicar si el programa está arrancado o detenido. Configurar el programa para que se inicie automáticamente con el ordenador como un servicio.

Este ejercicio proporcionará una comprensión práctica de cómo desarrollar scripts que operan de manera autónoma al iniciar el sistema, ofreciendo una valiosa experiencia en la implementación de servicios automatizados.

Estas serían las funciones que debe realizar el script (escrito en shell script importante).
En esta práctica se pondrán a prueba los conocimientos aprendidos en la anterior evaluación en comandos de linux y los aprendidos en la actual evaluación sobre shell script.
El servicio será lanzado en una máquina virtual con permisos de administrador ya que el usuario de “ugrupo1” no está en sudoers y por ende no tiene permisos de administración.

Objetivos

Introducción al Desarrollo de Servicios en Shell Script:

- Presentar el propósito de la práctica, centrado en el desarrollo de un script en Shell que funcionará como servicio al iniciar la máquina virtual.
- Ofrecer una visión general de los objetivos específicos que se abordarán en el documento.

2. **Funcionalidades del Programa en Shell Script:**
   - Describir detalladamente las funcionalidades del script, enfocándose en la recepción de argumentos como start, stop, restart, y status.
   - Explicar cómo el script manejará casos de error, como la ausencia de argumentos o la recepción de argumentos no esperados.
3. **Implementación de la Funcionalidad 'start' en Shell Script:**
   - Detallar el comportamiento del script cuando se recibe el argumento 'start'.
   - Explicar el mensaje a mostrar si el programa ya está arrancado y el proceso a seguir si no lo está, incluyendo la escritura en un archivo de registro y la emisión periódica del mensaje "Estoy vivo".
4. **Gestión de la Funcionalidad 'stop' en Shell Script:**
   - Definir la respuesta del script al argumento 'stop'.
   - Explicar el mensaje a mostrar si el programa está detenido y la acción a realizar si está arrancando, incluyendo la escritura en el registro y la detención de la ejecución.
5. **Implementación de la Funcionalidad 'restart' en Shell Script:**
   - Describir cómo el script llevará a cabo la funcionalidad 'restart', deteniendo y arrancando el programa de manera coherente.
6. **Funcionalidad 'status' en Shell Script:**
   - Definir la respuesta del script al argumento 'status'.
   - Describir cómo el script informará si está arrancado o detenido.
7. **Configuración del Script como Servicio:**
   - Presentar el objetivo final de configurar el script para que se inicie automáticamente con el ordenador como servicio.
   - Proporcionar pasos específicos para lograr esta configuración en entornos basados en Linux.

Shell Script

El código está escrito con un case que dispone de los casos: start,stop,restart y status.
Antes de proceder hacemos chmod 700 para solo poderlo ejecutar con el usuario root.
Abrimos con #!/bin/bash y definimos las 2 variables que vamos a ir utilizando en el programa, las cuales son $log y $pid. Cada una de ellas va asociada a un documento con su ruta correspondiente.

```
1 #!/bin/bash
2 log=/home/carlos/dimoni.log
3 pid=/home/carlos/dimoni.pid
4 case $1 in
5     start)
6         echo "start"
7         while true
8         do
9                 if [ -f $pid ]
10                then
11                        echo "el Programa ya estaba arrancado" >>$log
12                        echo "revisa el log, programa en curso" >>$log
13
14                else
15                         echo "Arrancando el programa" >>$log
16                         echo "estoy vivo" >>$log
17                         echo "$$" >>$pid #damos valor del id al proceso $pid
18                         sleep 2
19                fi
20        done
21        ;;
```

El código del programa es un case con las opciones de start, status, restart y stop.
En el case **start** tenemos un condicional. Si existe $pid va a mandar al log un echo con "el programa ya estaba arrancado". En caso contrario mandará un echo indicando que está arrancando el programa, y cada 2 segundos le dirá al log que sigue vivo y mandara el pid a $pid.

```
        ;;
    stop)
        if [ -f $pid ]
        then
            echo "Deteniendo el programa"
            programa=$(cat $pid)
            kill -9 $programa
            rm -f $pid
            echo "Programa detenido" >>$log
        else
            echo "El programa en cuestión ya estaba detenido" >>$log
                exit 4
        fi
        ;;
```

En el case **stop** es más de lo mismo. Si existe $pid va a mandar al log "deteniendo el programa" y va a matar $programa (que muestra el pid) más remover el fichero dimoni.pid ($pid) Una vez borrado mandaŕa al log que ya ha detenido el programa. En caso de que no exista $pid indicará que el programa ya estaba detenido.

```
33        ;;
34    restart)
35        $0 stop
36        $0 start &
37        ;;
38    status)
39        # Verifica si se proporciona un argumento
40        if [ "$1" == "" ] || [ "$1" == " " ]
41        then
42            echo "ERROR: Introduce un argumento" >>$log
43            exit 1
44        fi
45
46        while true
47        do
48            if [ -f $pid ]
49            then
50                echo "Comprueba el .log"
51                echo "El programa está arrancado" >>$log
52                sleep 1
53            else
54                 echo "Comprueba el log"
55                 echo "El programa no está arrancado" >>$log
56            fi
57            sleep 2
58        done
59        ;;
60    *)
61        echo "ERROR: Introduce un argumento válido" >>$log
62        exit 3
63        ;;
64 esac
```

El case **restart** detiene el programa y lo vuelve a arrancar en background (segundo plano).

Status comprueba que lo introducido sea un argumento. En caso afirmativo realiza un condicional, si existe $pid nos dice que el programa está arrancado y de lo contrario nos dice qué está detenido.

Para el resto de casos nos pedirá que introduzcas un argumento válid

Systemd

Para hacernos una idea de lo que es Systemd, Systemd es un directorio en el cual contiene todos los servicios del sistema.

Este directorio se encuentra en /etc/systemd/system/(Todos los servicios del sistema) para hacernos una idea más clara de lo importante que es esto, si nosotros queremos crear un servicio en cualquier directorio que no sea systemd/system, no te funcionara nunca.

Si nosotros queremos crear un servicio tenemos que tener en cuenta dos cosas, la primera es que el servicio sea creado en el directorio /etc/systemd/system, y la segunda es que el fichero termine en x".service"

Comandos que debes conocer para poder usar a la hora de modificar o actualizar o visualizar servicio del systemd.

- **systemctl status (Nombre del servicio)**

Con este comando sabemos el estado del servicio (Que estos pueden ser "Active", "Stop", o de algún error.

- **systemctl start (Nombre del servicio)**

Con este comando arrancamos el servicio.

- **systemctl stop (Nombre del servicio)**

Con este comando detenemos/paramos el servicio.

- **systemctl daemon-reload (Nombre del servicio)**

Esta orden nos permite recargar todos los servicios de nuevo.

- **systemctl restart (Nombre del servicio)**

Se utiliza para reiniciar un servicio o unidad específica gestionada por systemd.

- **systemctl kill (Nombre del servicio)**

Se utiliza para enviar una señal de término (o cualquier otra señal especificada) a un proceso o unidad gestionada por systemd.

Creación del Servicio

Para poder crear un servicio necesitamos primero saber, donde puedo crear uno y como debo de configurar, el directorio para poder crear servicios es el "/etc/systemd/system/".

Ahora para poder configurar/crear un servicio necesitaremos dividir el servicio en 3 secciones.

Y con algunas líneas de configuración en cada una de ellas

[Unit]
Proporciona información y configuración relacionada con la unidad de systemd a la que pertenece el servicio.

Description:
Proporciona una descripción humana del servicio o unidad. Es una descripción que ayuda a entender la función del servicio. No afecta el funcionamiento del servicio en sí, pero es útil para que los administradores del sistema comprendan su propósito.

After:
Especifica las unidades que deben iniciarse antes de la unidad actual. En otras palabras, indica las dependencias de la unidad. Si una unidad tiene After=network.target, por ejemplo, significa que esta unidad se iniciará después de que la red esté completamente activa. Lo mismo pasa con before, pero con el efecto contrario.

[Service]
Contiene la configuración específica del servicio que se está definiendo.

User y Group:
Especifica el usuario y el grupo bajo el cual se ejecutará el servicio. Puedes definir estos valores para restringir los privilegios del servicio y mejorar la seguridad.

ExecStart:
Indica el comando o script que se ejecutará para iniciar el servicio. Puedes usar esta opción para especificar la ruta completa al ejecutable o script que inicia tu servicio.

WorkingDirectory:
Establece el directorio de trabajo para el servicio. Esto es útil si tu servicio depende de archivos en un directorio específico o si necesita ejecutarse desde un directorio en particular.

[Install]
Se utiliza para definir cómo el servicio debería ser instalado en el sistema y habilitado para que se inicie automáticamente.

WantedBy:
Indica la meta (target) que solicitará la activación de esta unidad. Generalmente, se establece en una de las metas de nivel de ejecución, como multi-user.target para sistemas multiusuario o graphical.target para sistemas con interfaz gráfica.

[Unit]
Descripción: Proporciona una descripción del servicio, que es útil para entender su propósito.

[Service]
User: Especifica el nombre de usuario bajo el cual se ejecutará el servicio. En este caso, el servicio se ejecutará como el usuario victor.

ExecStart: Especifica el comando o script que se ejecutará para iniciar el servicio. Aquí se utiliza PHP para ejecutar el script /usr/local/bin/demonio.sh.

[Install]
WantedBy: Específica la meta (o nivel de ejecución) en el cual el servicio debe ser habilitado. En este caso, se habilita en multi-user.target, que es el nivel de ejecución que se utiliza para sistemas multiusuario.

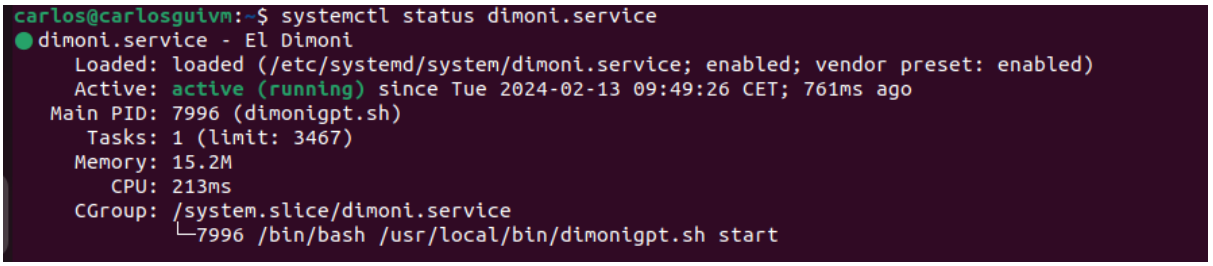
```
[Unit]
Description=El Dimoni
After=network.target

[Service]
#User=carlos
ExecStart=/usr/local/bin/dimoni.sh start
#Type=simple
#RemainAfterExit=yes
#WorkingDirectory=/usr/local/bin/dimoni.sh
#Restart=always

[Install]
WantedBy=multi-user.target
```

En este caso, la ruta es /usr/local/bin/dimoni.sh y añadimos el start para indicar que al encender el equipo se inicie automáticamente.
Tenemos texto comentado ya que previamente hice pruebas y no me funcionaba como quería así que lo comenté para que no afectará al funcionamiento del servicio.

Una vez creado y configurado el .service, tocará habilitarlo con: sudo systemctl enable dimoni.service y luego hacer un systemctl daemon-reload para reiniciar los servicios.

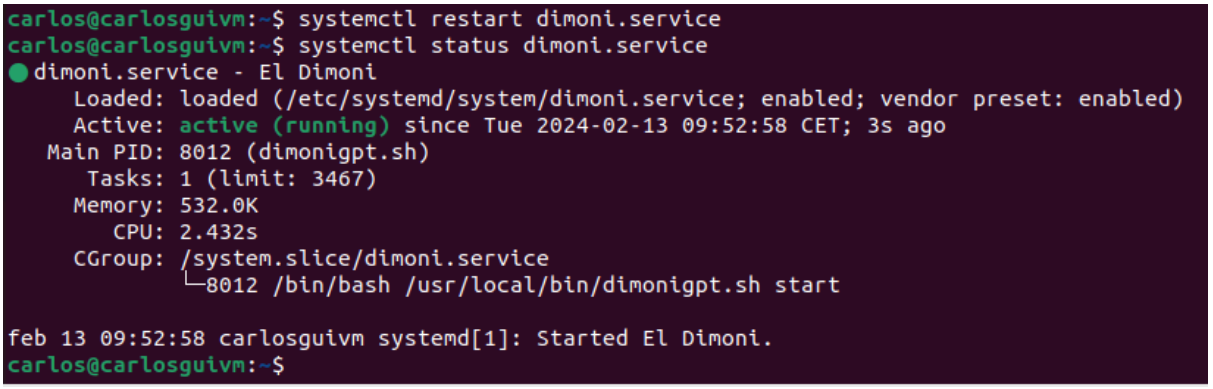
Pruebas de uso

Es probable que encuentres varios errores durante las pruebas de uso, ya sea por descuidos o pequeños detalles, como olvidar mover el script a la ruta correcta. También puede suceder que haya errores en el código que solo notes al final.



Para abordar esto, al principio es probable que surjan errores. Una solución sencilla consiste en reiniciar todos los servicios mediante el comando "systemctl daemon-reload". Luego, para verificar el estado del servicio, utiliza "systemctl status (nombre del servicio)".

Es posible que al principio experimentes errores si olvidas mover el script a su directorio correspondiente. Para solucionarlo, ejecuta el comando "mv ./(directorio actual/correspondiente) /usr/local/bin/" para trasladar el script al directorio designado para la ejecución de scripts.



Conclusiones

Con esta práctica me he quemado la cabeza constantemente porque no me funcionaba nada a la primera. Tarde más de una semana en que me funcionará todo decentemente.

Ha sido un acierto al 100% realizar la práctica con más de 2 semanas de antelación porque si no el nivel de frustración habría sido mucho peor. He aprendido una barbaridad y adquirido fluidez a la hora de codificar, sin contar que soy mucho más limpio a la hora de escribir el código.

Para el servicio estaba perdidisimo, he tenido que buscar en internet y preguntado a varios compañeros porque me daba mucho error (luego me dí cuenta que el problema era que no estaba modificando el fichero correcto XD)

En conclusión está práctica ha sido un reto divertido pero me ha dado más problemas de los que debería. Me sigue sin gustar la programación, pero es muy satisfactorio que tus programas funcionen y definitivamente notar que vas mejorando es muy guay.

Anexo 1

```
#!/bin/bash
log=/home/carlos/dimoni.log
pid=/home/carlos/dimoni.pid
case $1 in
   start)
       while true
       do
       echo "start"

       if [ -f $pid ]
       then
                    echo "el Programa ya estaba arrancado"
                    echo "revisa el log, programa en curso"

       else
                     echo "Arrancando el programa"
                     echo "estoy vivo" >>$log
                     echo "$$" >>$pid #damos valor del id al proceso $pid
                     sleep 2
       fi
      done
      ;;
   stop)
       if [ -f $pid ]
       then
          echo "Deteniendo el programa"
          programa=$(cat $pid)
          kill -9 $programa
           rm -f $pid
          echo "Programa detenido" >>$log
```

```
        else
            echo "El programa en cuestión ya estaba detenido" >>$log
         exit 4
        fi
        ;;
    restart)
        $0 stop
        $0 start &
        ;;
    status)
        # Verifica si se proporciona un argumento
        if [ "$1" == "" ] || [ "$1" == " " ]
         then
            echo "ERROR: Introduce un argumento" >>$log
            exit 1
        fi

        while true
        do
            if [ -f $pid ]
            then
                echo "Comprueba el .log"
                echo "El programa está arrancado" >>$log
                sleep 1
            else
                    echo "Comprueba el log"
                    echo "El programa no está arrancado" >>$log
            fi
            sleep 2
        done
        ;;
    *)
        echo "ERROR: Introduce un argumento válido" >>$log
        exit 3
        ;;
esac
```

Anexo 2

He tenido muchos problemas con el código en sí ya que al encender el servicio no se ejecutaba el script, y eso era pq estaba directamente mal codificado ya que no salía del primer bucle (un while true concretamente) y no leía el resto de casos. Me di cuenta tras ver que en el log solo el mensaje de "estoy vivo" se repetía infinidad de veces.

La solución fue transformar el while y moverlo directamente al case start (estaba ejecutándose fuera del case y por ende este último no se ejecutaba).

Luego tuve un problema con el servicio que no funcionaba bien. Resulta que todo el fichero que estaba modificando era otro y por eso daba igual lo que tocara no cambiaba la situación, me di cuenta al mirar el código de error. La solución fue reemplazar el archivo con el que estaba modificando y con eso logré que funcionara el servicio.

Tuve luego otro problema pero esta vez al realizar las comprobaciones finales para este documento. En esencia, $pid ni se creaba ni se borraba. Lo solucione rehaciendo ambos casos.

Webgrafía

https://systemd.io/
https://www.redeszone.net/tutoriales/servidores/administrar-servicios-linux-systemd/