



PROYECTO INTEGRADOR: ○ GESTOR AUTOMATIZADO DE SERVICIOS CON NOTIFICACIÓN POR TELEGRAM

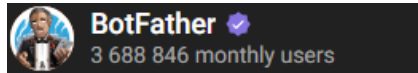
MANUAL TECNICO

Christopher Gomez Mendoza
Jesus Valentin Mora Cordoba
Irvin Josafat Dominguez Morales
Mario Erik Flandes Hernandez

Lic. Ingeniería en Ciberseguridad e
Infraestructura de Computo.
Programación en la Administración de
Servicios

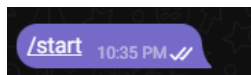
Creación del BOT:

En este apartado se explicará los pasos para la creación del Bot de telegram con BotFather

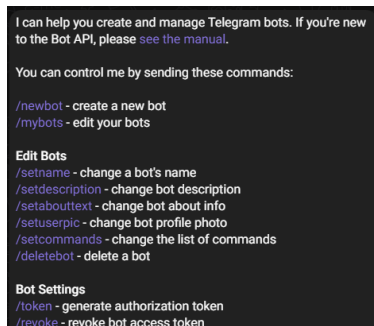


BotFather es el Bot oficial de Telegram para crear y gestionar otros bots.

Iniciar conversación con BotFather



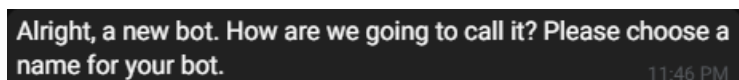
Pulsa Iniciar /start



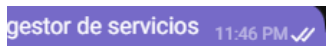
Se mostrará un menú con las opciones que nos proporcionará BotFather

Crearemos un nuevo Bot

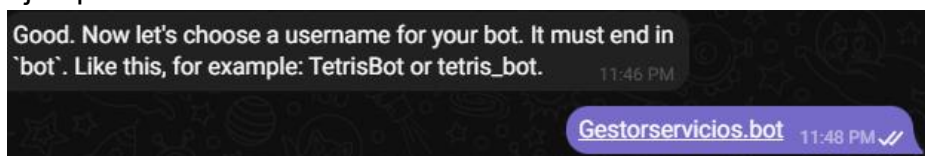
/newbot



“Muy bien, un nuevo bot. ¿Cómo lo vamos a llamar? Por favor elige un nombre para tu bot.”



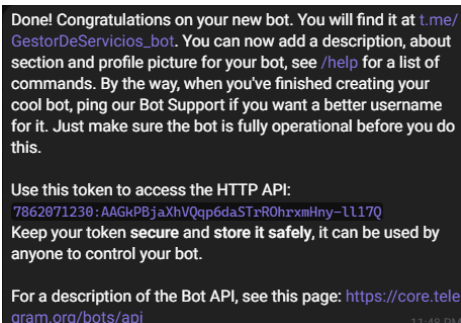
“Bien. Ahora elige un nombre de usuario para tu bot. Debe terminar en bot, como por ejemplo: TetrisBot o tetrís_bot.”



BotFather confirma la creación del bot y te proporciona:

El enlace para acceder a tu bot: t.me/GestorDeServicios_bot

El token de autenticación:
7862071230:AAGkPBjaXhVQqp6daSTrR0hrxxmHny-LL17Q



Done! Congratulations on your new bot. You will find it at t.me/GestorDeServicios_bot. You can now add a description, about section and profile picture for your bot, see [/help](#) for a list of commands. By the way, when you've finished creating your cool bot, ping our Bot Support if you want a better username for it. Just make sure the bot is fully operational before you do this.

Use this token to access the HTTP API:
`7862071230:AAGkPBjaXhVQqp6daSTrR0hrxxmHny-LL17Q`
Keep your token secure and store it safely, it can be used by anyone to control your bot.

For a description of the Bot API, see this page: <https://core.telegram.org/bots/api>

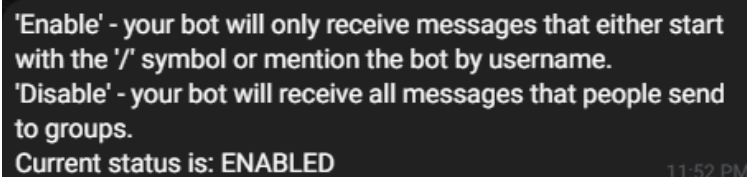
“¡Listo! Felicitaciones por tu nuevo bot. Puedes agregar una descripción, imagen de perfil, etc. Usa este token para acceder a la API HTTP. Guarda el token de forma segura.”

Estado del Bot

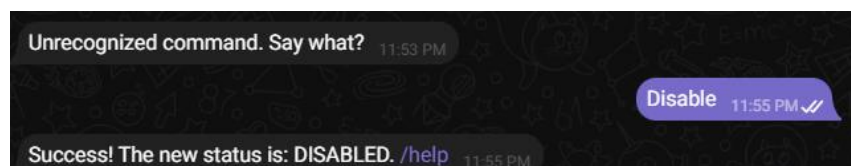
Puedes elegir cómo tu bot responderá en grupos:

Enable: Solo responderá si lo mencionas o si se usa `/comando`.

Disable: Recibirá todos los mensajes que se envían en el grupo.



'Enable' - your bot will only receive messages that either start with the '/' symbol or mention the bot by username.
'Disable' - your bot will receive all messages that people send to groups.
Current status is: ENABLED

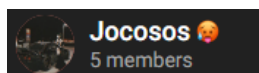


Unrecognized command. Say what? 11:53 PM

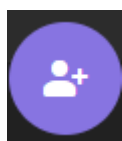
Disable 11:55 PM ✓

Success! The new status is: DISABLED. [/help](#) 11:55 PM

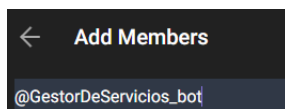
Agregar el nuevo bot a nuestro grupo



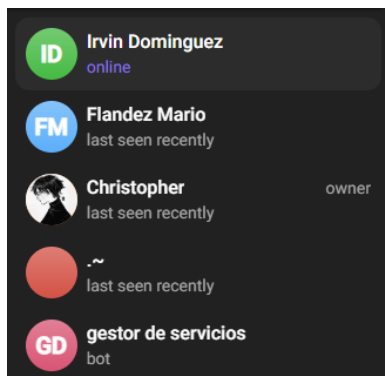
En nuestro grupo le daremos en el botón de añadir nuevo usuario



Y buscaremos con el nombre de usuario que le proporcionamos



Y listo



Ya está el Bot para la chamba

Usuarios.sh

Es un código realizado en Bash que es crear un bot que funcione en Telegram y que permita poder gestionar usuarios. Que unas cosas de esas eran poder crear nuevos usuarios, modificar nombres de los usuarios y poder eliminar usuarios existentes. Mediante un menú que se envía.

Lo que incluye el código es:

- bot.sh: Es el script que escucha los mensajes de Telegram
- usuarios.sh: Es el script que hace las acciones pedidas que son las de crear, modificar y eliminar.
- Config.txt: Es la configuración del TOKEN y del CHAT_ID
- Usuarios.log: Es donde se guardan todos los registros realizados con el bot hasta errores.

Como el siguiente script que se muestra es el de usuarios.sh

```
#!/bin/bash

# Se declaran las variables de configuración de nombre para no tener conflictos al momento de utilizarlas
archivo_config="config.txt" # Es el archivo donde está guardada la configuración del TOKEN y del CHAT_ID
archivo_log="usuarios.log" # Es donde se van a guardar los registros de todas las acciones que se hagan con el bot

# Es donde se guardan las acciones que utilizan como punto de crear, modificar y eliminar
crear="cr" # Es donde se guarda el nombre del usuario ingresado para hacer las acciones
eliminar="el" # Es donde se guarda el nombre del usuario ingresado para hacer las acciones
modificar="mo" # Es donde se guarda el nombre del usuario ingresado para hacer las acciones

# Es para poder seleccionar el archivo desde esta configuración y ejecutar lo que contiene como si fuera un if [[ "accion" ]] then
if [[ "$accion" == "crear" ]] then
    mensaje="No se ingresó el nombre para modificar el usuario"
    elif echo nombre "$accion" | grep -q "^[a-zA-Z0-9_]+$"; then
        mensaje="Usuario $accion creado correctamente" # Se crea un nuevo usuario
    else
        mensaje="No se pudo crear el usuario: $accion" # No existía el usuario
    fi
else
    mensaje="La acción $accion no es válida"
fi

# Es el código que hace la función de poder crear
if [[ "$accion" == "crear" ]] then
    if echo nombre "$accion" | grep -q "^[a-zA-Z0-9_]+$"; then
        mensaje="Usuario $accion creado correctamente" # Se crea un nuevo usuario
    else
        mensaje="No se pudo crear el usuario: $accion" # No existía el usuario
    fi
fi

# Es el código que hace la función de poder eliminar
if [[ "$accion" == "eliminar" ]] then
    if echo nombre "$accion" | grep -q "^[a-zA-Z0-9_]+$"; then
        mensaje="Usuario $accion eliminado correctamente"
    else
        mensaje="No se pudo eliminar el usuario: $accion"
    fi
fi

# Es el código que hace la función de poder modificar
if [[ "$accion" == "modificar" ]] then
    if echo nombre "$accion" | grep -q "^[a-zA-Z0-9_]+$"; then
        mensaje="No se ingresó el nombre para modificar el usuario"
        elif echo nombre "$accion" | grep -q "^[a-zA-Z0-9_]+$"; then
            mensaje="Usuario $accion modificado a: $accion"
        else
            mensaje="No se pudo modificar el usuario: $accion"
        fi
    else
        mensaje="La acción $accion no es válida" # En caso de que selecciono de manera incorrecta algo arroja esto
    fi
fi

# Es donde se guardan y muestran los resultados que se obtuvieron
echo mensaje "$mensaje"
guardar_log "$mensaje"
```

El siguiente es el de bot.sh

[illegible]

```

40 elif [[ "$accion" == "eliminar" ]]; then
41     estado=$(cat "$archivo_estado") || { echo "Archivo no encontrado"; exit 1; }
42     echo "Eliminando mensaje..."
43     curl -X POST "https://api.telegram.org/bot$token/sendMessage" \
44     -d "chat_id=$chat_id" \
45     -d "text=Se eliminó el mensaje."
46     echo "Mensaje eliminado."
47 fi
48
49 # Actualizar estado
50 estado=$(cat "$archivo_estado") || { echo "Archivo no encontrado"; exit 1; }
51 echo "Actualizando estado..."
52 curl -X POST "https://api.telegram.org/bot$token/sendMessage" \
53     -d "chat_id=$chat_id" \
54     -d "text=Estado actualizado."
55     echo "Estado actualizado."
56 fi
57
58 # Manejar comandos de chat
59 while true; do
60     read -r chat_id chat_text
61     if [[ "$chat_text" == "/start" ]]; then
62         echo "Comando /start recibido."
63         curl -X POST "https://api.telegram.org/bot$token/sendMessage" \
64             -d "chat_id=$chat_id" \
65             -d "text=¡Hola! Soy un bot de Telegram."
66     elif [[ "$chat_text" == "/estado" ]]; then
67         echo "Comando /estado recibido."
68         estado=$(cat "$archivo_estado") || { echo "Archivo no encontrado"; exit 1; }
69         echo "Estado actual: $estado"
70     elif [[ "$chat_text" == "/eliminar" ]]; then
71         echo "Comando /eliminar recibido."
72         estado=$(cat "$archivo_estado") || { echo "Archivo no encontrado"; exit 1; }
73         echo "Eliminando mensaje..."
74         curl -X POST "https://api.telegram.org/bot$token/sendMessage" \
75             -d "chat_id=$chat_id" \
76             -d "text=Se eliminó el mensaje."
77         echo "Mensaje eliminado."
78     fi
79 done

```

Luego pide por mensaje que se ingrese el dato necesario.

4

Instalaciones que se necesitaron hacer fueron:

- Curl Se usa para enviar mensajes y leer respuestas desde la API de Telegram.
- sudo apt install curl
- jq Se usa para leer y analizar los datos en formato JSON (mensajes, botones, etc.).
- sudo apt install jq
- sudo Permite ejecutar comandos como useradd, userdel, usermod.

Este es el resultado de como se ve el bot en el chat de telegram haciendo las pruebas:



Estos son los registros que fueron guardados en el archivo de usuarios.log:

```
2025-06-23 00:48:04 - No se pudo crear el usuario gallo
2025-06-23 00:48:16 - No se pudo eliminar el usuario gallo
2025-06-23 00:48:46 - No se pudo cambiar el nombre del usuario gallo
2025-06-23 00:53:56 - Usuario miyagi creado correctamente
2025-06-23 00:54:21 - Usuario miyagi cambiado a irvin
2025-06-23 00:54:33 - Usuario irvin eliminado correctamente
2025-06-23 13:10:08 - Usuario: Valentin creado correctamente
2025-06-23 13:10:56 - Usuario Valentin cambio a: chalino
2025-06-23 13:11:16 - Usuario: chalino se elimino correctamente
```

Respaldo.sh

Este script se encarga de crear un respaldo comprimido (.tar.gz) de un directorio específico, cuyo valor se define en el archivo externo "config.txt". El respaldo se guarda en un directorio definido y se notifica automáticamente vía Telegram si el respaldo fue exitoso o fallido.

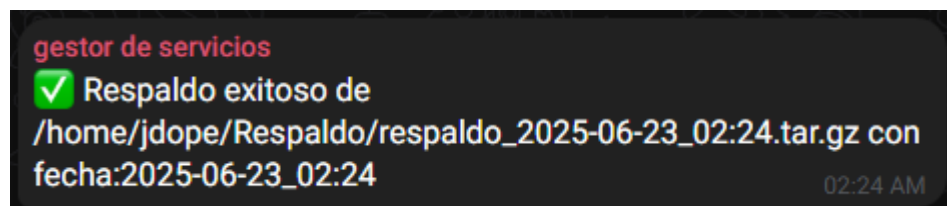
Respaldo.sh

```
1  #!/bin/bash
2
3  #Este script se encarga de comprimir un directorio y notificar por Telegram
4
5  #Establecemos PATH para el uso correcto de cron
6  export PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
7  #Cargamos las configuraciones
8  source /home/jdope/Proyecto_Adm.Servicios/Proyecto-Prograservicios/config.txt
9
10 #Convertimos la fecha y el archivo respaldo en variables para resumirlos y que sea mas facil usarlos
11 Fecha=$(date +%Y-%m-%d_%H:%M)
12 Archivo_respaldado="$Directorio_respaldado/respaldo_$Fecha.tar.gz"
13
14 #Debugeo de rutas (para probar que sean obtenidas del config.txt correctamente)
15 echo "Directorio a respaldar [$Directorio_a_respaldar]"
16 echo "Directorio respaldado [$Directorio_respaldado]"
17 echo "Archivo resultante: [$Archivo_respaldado]"
18
19 #En caso de que no exista el directorio destino, esta linea se encarga de crearlo para evitar errores en caso de
20 #que no se haya definido correctamente en config.txt
21 mkdir -p "$Directorio_respaldado"
22
23 #Se crea el respaldo
24 echo "Creando el respaldo..."
25 tar -czf "$Archivo_respaldado" "$Directorio_a_respaldar"
26
27 #if para verificar si se hizo el respaldo correctamente
28 if [[ -f "$Archivo_respaldado" ]]; then
29     Mensaje_para_bot="✅ Respaldo exitoso de $Archivo_respaldado con fecha:$Fecha"
30 else
31     Mensaje_para_bot="⚠ Error: Ocurrio un fallo al crear el respaldo"
32 fi
33
34 # Enviar notificación
35 /usr/bin/curl -s -X POST "https://api.telegram.org/bot$TOKEN/sendMessage" \
36     -d chat_id="$CHAT_ID" \
37     -d text="$Mensaje_para_bot"
38
39 #Bitacora al .log
40 echo "[$(date +%Y-%m-%d_%H:%M)] Respaldo $Directorio_a_respaldar -> $Archivo_respaldado" >> /home/jdope/respaldo.log
```

Salida en consola al ejecutar el script

```
root@basadope:/home/jdope/Proyecto_Adm.Servicios/Proyecto-Prograservicios# ./respaldo.sh
Directorio a respaldar [/home/jdope/Documentos]
Directorio respaldado [/home/jdope/Respaldo]
Archivo resultante: [/home/jdope/Respaldo/respaldo_2025-06-23_02:24.tar.gz]
Creando el respaldo...
tar: Eliminando la '/' inicial de los nombres
{"ok":true,"result":{"message_id":799,"from":{"id":7862071230,"is_bot":true,"first_name":"gestor de servicios","username":"GestorDeServicios_bot"},"chat":{"id":-1002891538692,"title":"Jocosos \ud83e\udd75","type":"supergroup"},"date":1750667050,"text":"\u2705 Respaldo exitoso de /home/jdope/Respaldo/respaldo_2025-06-23_02:24.tar.gz con fecha:2025-06-23_02:24"}}root@basadope:/home/jdope/Proyecto_Adm.Servicios/Proyecto-Prograservicios#
```

Mensaje de autenticación al bot de Telegram.



Carpeta de Respaldo y todas las copias de seguridad hechas

```
root@basadope:/home/jdope/Respaldo# ls -l
total 52
-rw-r--r-- 1 root root 464 jun 21 19:54 respaldo_2025-06-21_19-54.tar.gz
-rw-r--r-- 1 root root 464 jun 21 19:59 respaldo_2025-06-21_19-59.tar.gz
-rw-r--r-- 1 root root 464 jun 21 20:08 respaldo_2025-06-21_20-08.tar.gz
-rw-r--r-- 1 root root 464 jun 21 20:09 respaldo_2025-06-21_20-09.tar.gz
-rw-r--r-- 1 root root 464 jun 21 20:13 respaldo_2025-06-21_20-13.tar.gz
-rw-r--r-- 1 root root 464 jun 21 20:22 respaldo_2025-06-21_20-22.tar.gz
-rw-r--r-- 1 root root 464 jun 21 20:23 respaldo_2025-06-21_20-23.tar.gz
-rw-r--r-- 1 root root 464 jun 23 01:00 respaldo_2025-06-23_01-00.tar.gz
-rw-r--r-- 1 root root 464 jun 23 02:24 respaldo_2025-06-23_02-24.tar.gz
-rw-r--r-- 1 root root 464 jun 23 02:25 respaldo_2025-06-23_02-25.tar.gz
-rw-r--r-- 1 root root 464 jun 23 02:26 respaldo_2025-06-23_02-26.tar.gz
-rw-r--r-- 1 root root 464 jun 23 02:27 respaldo_2025-06-23_02-27.tar.gz
-rw-r--r-- 1 root root 464 jun 23 02:28 respaldo_2025-06-23_02-28.tar.gz
root@basadope:/home/jdope/Respaldo#
```

Monitoreo.sh

Este script se encarga de leer el porcentaje del uso de CPU y espacio en disco, mandando una alerta por Telegram si se alcanza el límite por defecto estará en 70% cada uno, pudiendo modificar los límites mediante mensajes de Telegram.


```

1 #!/bin/bash
2
3 source config.txt # Carga el archivo donde tiene los parametros reutilizables
4
5 if [[ -f offset.txt && -s offset.txt ]]; then # Verificamos que el archivo offset exista y no este vacio si es asi lee el contenido
6     idSiguiente=$(cat offset.txt) # del archivo, si no se cumple alguna se le asigna el valor de 0
7 else
8     idSiguiente=0
9 fi
10
11 respuesta=$(curl -s "https://api.telegram.org/bot$TOKEN/getUpdates?offset=$idSiguiente") # Le pedimos a Telegram los mensajes no leidos a
12 # partir de que mensaje
13
14
15 mensaje=$(echo "$respuesta" | grep -o '"text": "[^"]+"' | cut -d ':' -f2 | tr -d '"') # Extraemos el texto del ultimo mensaje
16
17 ultimo_id=$(echo "$respuesta" | grep -o '"update_id": [0-9]*' | tail -1 | cut -d ':' -f2) # Extraemos el update_id del mensaje y lo guardamos
18 if [[ -n "$ultimo_id" ]]; then # en el archivo offset para que no se repitan los
19     echo $((ultimo_id + 1)) > offset.txt # mensajes cada que se ejecute el script
20 fi
21
22 url="https://api.telegram.org/bot$TOKEN/sendMessage" # Variable del url del bot
23
24
25 if [[ "$mensaje" == /cpu* ]]; then
26     valor=$(echo "$mensaje" | awk '{print $2}') # Condicionales para comprobar si el mensaje
27     # recibido es /cpu o /disco y cambiar el valor
28     sed -i "s/~limiteCpu~/limiteCpu=$valor/" config.txt # en el archivo config.txt y mandar un mensaje de
29     # que se cambio el limite del cpu o disco respectivamente
30 fi
31
32 if [[ "$mensaje" == /disco* ]]; then
33     valor=$(echo "$mensaje" | awk '{print $2}')
34     sed -i "s/~limiteDisco~/limiteDisco=$valor/" config.txt
35     curl -s -X POST $url -d chat_id="$CHAT_ID" -d text="~ Limite de Disco cambiado a $valor~"
36 fi
37
38 source config.txt # Cargamos el archivo con los nuevos valores
39
40
41 cpu=$(mpstat 1 1 | awk '/Media/ {print 100 - $0}' | cut -d '.' -f1) # Obtenemos el numero del cpu utilizado
42 disco=$(df / | tail -1 | awk '{print $5}' | tr -d '%') # Obtenemos el espacio ocupado del disco
43
44 echo "🚀 Valor de CPU medido: $cpu%"
45
46 if [[ "$mensaje" == /estado* ]]; then # Condicional para ver si el ultimo mensaje es /estado, si es manda el mensaje con los valores
47     curl -s -X POST $url -d chat_id="$CHAT_ID" -d text="📊 Estado del sistema: 🟢 CPU: $cpu% 🟢 Disco: $disco%"
48 fi
49
50 alerta="" # Inicializamos alerta vacio
51
52 if (( cpu > limiteCpu )); then # Si cpu es mayor que el limite del cpu guarda en alerta el mensaje de cpu alta
53     alerta="CPU muy alta: $cpu% --> Limite $limiteCpu%"
54 fi
55
56 if (( disco > limiteDisco )); then # Si disco es mayor que el limite del disco guarda en alerta el mensaje de disco lleno
57     alerta="Disco casi lleno: $disco% --> Limite $limiteDisco%"
58 fi
59
60 mensaje=$(echo -e "📊 Estado del sistema:\n🟢 CPU: $cpu% --> limite $limiteCpu%\n🟢 Disco: $disco% --> limite $limiteDisco%\n🔔 Alertas:\n$alerta")
61
62 if [[ -n "$alerta" ]]; then
63     curl -s -X POST $url -d chat_id="$CHAT_ID" -d text="$mensaje" # Si alerta no esta vacia entonces manda el mensaje al bot
64 fi
65
66

```

Paquetes instalados curl para conectar con la Api de Telegram, mpstat para la medición de CPU y stress para hacer las pruebas.

Sudo apt install curl

Sudo apt install sysstat

Sudo apt install stress

Línea 1-10

Carga la configuración del archivo config.txt, y lee el ultimo id + 1 del mensaje de Telegram.

Línea 11-20

Obtenemos el mensaje de telegram y guardamos el id del mensaje + 1 en el archivo offset.txt.

línea 22-35

Definimos la url para mandar los mensajes y ponemos condicionales para recibir los mensajes /cpu y /disco, y mandamos el mensaje de confirmación.

línea 38-47

Cargamos nuevamente el archivo config.txt con los datos actualizados y sacamos el valor actual de cpu y disco ocupado. Y agregamos el condicional que manda el mensaje de estado.

línea 50-58

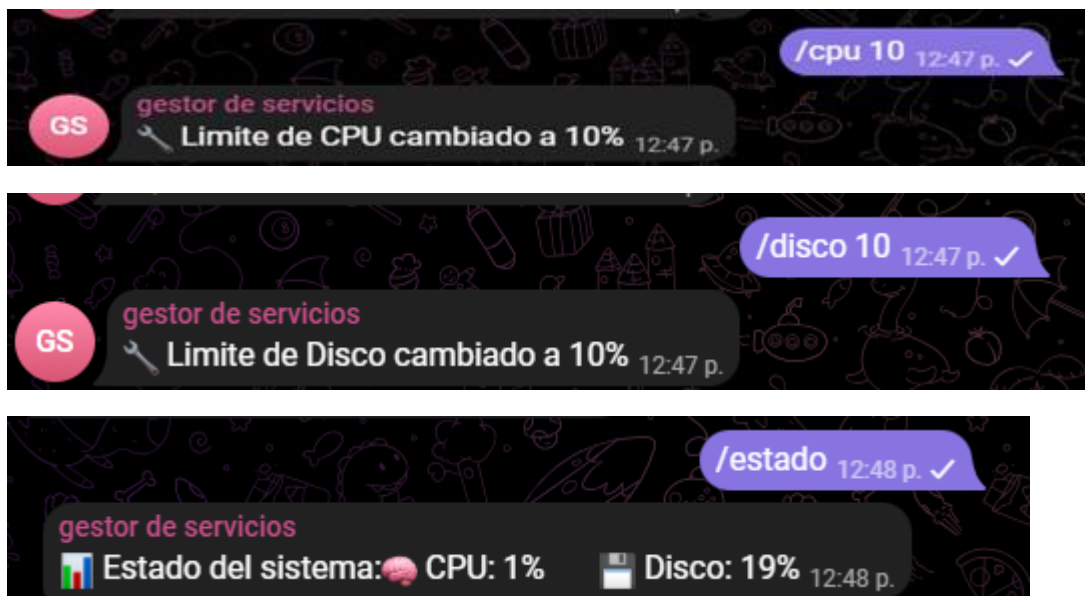
Evalúa los límites y genera las alertas.

línea 60-64

Envía mensaje si hay alertas.

Pruebas

envió de mensajes /cpu, /disco y estado desde Telegram.



Simulación de Alertas con stress y cambiando limite a disco

```
$ stress --cpu 2 --timeout 20
stress: info: [4480] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
stress: info: [4480] successful run completed in 20s
```

Estado del sistema:

- CPU: 100% → límite 10%
- Disco: 19% → límite 10%

Alertas:

CPU muy alta: 100% (máx 10%)

/disco 15 12:53 p. ✓

gestor de servicios

Límite de Disco cambiado a 15% 12:53 p.

Estado del sistema:

- CPU: 1% → límite 70%
- Disco: 19% → límite 15%

Alertas:

Disco casi lleno: 19% → Límite 15% 12:53 p.

GS

```

1 #!/bin/bash
2
3 source config.txt # Carga el archivo donde tiene los parametros reutilizables
4
5 if [[ -f offset.txt && -s offset.txt ]]; then # Verificamos que el archivo offset exista y no este vacio si es asi lee el contenido
6     idSiguiente=$(cat offset.txt) # del archivo, si no se cumple alguna se le asigna el valor de 0
7 else
8     idSiguiente=0
9 fi
10
11 respuesta=$(curl -s "https://api.telegram.org/bot$TOKEN/getUpdates?offset=$idSiguiente") # Le pedimos a Telegram los mensajes no leídos a
12 # partir de que mensaje
13
14 mensaje=$(echo "$respuesta" | grep -o '"text": "[^"]*" ' | cut -d' ' -f2 | tr -d '"') # Extraemos el texto del ultimo mensaje
15
16 ultimo_id=$(echo "$respuesta" | grep -o '"update_id": [0-9]* ' | tail -1 | cut -d' ' -f2) # Extraemos el update_id del mensaje y lo guardamos
17 # en el archivo offset para que no se repitan los
18 if [[ -n "$ultimo_id" ]]; then
19     echo $((ultimo_id + 1)) > offset.txt # mensajes cada que se ejecute el script
20 fi
21
22 url="https://api.telegram.org/bot$TOKEN/sendMessage" # Variable del url del bot
23
24
25 if [[ "$mensaje" == /cpu* ]]; then
26     valor=$(echo "$mensaje" | awk '{print $2}') # Condicionales para comprobar si el mensaje
27     sed -i "s/^limiteCpu=./limiteCpu=$valor/" config.txt # recibido es /cpu o /disco y cambiar el valor
28     curl -s -X POST $url -d chat_id="$CHAT_ID" -d text="Limite de CPU cambiado a $valor%" # en el archivo config.txt y mandar un mensaje de
29 # que se cambio el limite del cpu o disco respectivamente
30 fi
31
32 if [[ "$mensaje" == /disco* ]]; then
33     valor=$(echo "$mensaje" | awk '{print $2}') # Condicionales para comprobar si el mensaje
34     sed -i "s/^limiteDisco=./limiteDisco=$valor/" config.txt # recibido es /cpu o /disco y cambiar el valor
35     curl -s -X POST $url -d chat_id="$CHAT_ID" -d text="Limite de Disco cambiado a $valor%" # en el archivo config.txt y mandar un mensaje de
36 # que se cambio el limite del cpu o disco respectivamente
37 fi

```

```

38 source config.txt # Cargamos el archivo con los nuevos valores
39
40 cpu=$(mpstat 1 1 | awk '/Media/ {print 100 - $NF}' | cut -d'.' -f1) # Obtenemos el numero del cpu utilizado
41 disco=$(df / | tail -1 | awk '{print $5}' | tr -d '%') # Obtenemos el espacio ocupado del disco
42
43 echo "📊 Valor de CPU medido: $cpu%"
44
45 if [[ "$mensaje" == /estado* ]]; then # Condicional para ver si el ultimo mensaje es /estado, si es manda el mensaje con los valores
46 | curl -s -X POST $url -d chat_id="$CHAT_ID" -d text="📊 Estado del sistema:\n📊 CPU: $cpu%\n📊 Disco: $disco%"
47 fi
48
49
50 alerta="" # Inicializamos alerta vacio
51
52 if (( cpu > limiteCpu )); then # Si cpu es mayor que el limite del cpu guarda en alerta el mensaje de cpu alta
53 | alerta+="CPU muy alta: $cpu% (máx $limiteCpu%)\n"
54 fi
55
56 if (( disco > limiteDisco )); then # Si disco es mayor que el limite del disco guarda en alerta el mensaje de disco lleno
57 | alerta+="Disco casi lleno: $disco% (máx $limiteDisco%)\n"
58 fi
59
60 mensaje=$(echo -e "\n📊 Estado del sistema:\n📊 CPU: $cpu% (límite $limiteCpu%)\n📊 Disco: $disco% (límite $limiteDisco%)\n\n🚨 Alertas:\n$alerta")
61
62 if [[ -n "$alerta" ]]; then
63 | curl -s -X POST "$url" -d chat_id="$CHAT_ID" -d text="$mensaje" # Si alerta no esta vacia entonces manda el mensaje al bot
64 fi
65
66

```

Servicios.sh

El script servicios.sh tiene como objetivo automatizar la supervisión del estado de servicios esenciales del sistema como el (ssh, nginx, cron). Este script revisa si cada servicio está activo, y en caso de que alguno este detenido o caído, intentara reiniciarlo automáticamente. Cuando esto llegue ocurrir o detectarse, el sistema notificara al usuario, mediante un Bot creado en telegram un mensaje en tiempo real.

Sus propósitos son los siguientes:

- Debe revisar el estado de servicios definidos en una lista.
- Si un servicio está inactivo, debe intentar iniciarlo.
- Debe registrar los servicios caídos e informar por Telegram

Instalación

En esta sección se dará a conocer los comandos que se utilizaron para la instalación de los servicios a trabajar.

- apt install ssh

```
irvin@debian:~$ sudo apt install ssh
```

- apt install cron

```
irvin@debian:~$ sudo systemctl stop cron
```

- apt install nginx

```
irvin@debian:~$ sudo systemctl stop nginx_
```

Este script realiza el monitoreo automático de servicios del sistema definidos en un archivo de configuración (config.txt), intenta reiniciarlos si están caídos, envía mensajes a Telegram y guarda registros en un .logs.

- servicios.sh: Script principal que se ejecuta cada cierto tiempo.

```
GNU nano 7.2                                servicios.sh *
1  #!/bin/bash
2
3  # Carga las variables del archivo config.txt
4  source "$(dirname "$0")/config.txt" # dirname : obtienen la ruta donde este ubicado el archivo
5
6  mensaje_telegram() { #funcion para enviar las notificaciones
7      local mensaje="$1" # recibe como un parametro
8
9      # chat_id: del grupo donde se encuentra el bot y el mensaje que se ara llegar a telegram
10     curl -s -X POST "https://api.telegram.org/bot$TOKEN/sendMessage" \
11         -d chat_id="$CHAT_ID" \
12         -d text="$mensaje"
13 }
14 # Butle principal
15 for servicio in "${SERVICIOS[@]}; do #se recornera los servicios en la lista, que definimos en la seccion de SERVICIOS en config.txt
16     systemctl is-active --quiet $servicio # se verificara si el servicio esta activo
17
18     if [ $? -ne 0 ]; then # si esta inactivo (codigo diferente de 0) se intentara reiniciar
19         systemctl restart "$servicio" # reinicia el servicio, si c cumple la condicion
20         sleep 1 # espera 1 segundo para darle tiempo al servicio
21
22         systemctl is-active --quiet "$servicio" # vuelve a verificar si el servicio esta activo ( se reinicio automaticamente)
23
24         if [ $? -eq 0 ]; then # condicion donde el servicio se reinicio automaticamente ( codigo mayor a 0 ) se reinicio exitosamente
25             mensaje="onni-chan, el servicio $servicio se detubo o se cayo, pero se reinicio correctamente, el reinicio se realizo el dia $(date +%Y-%m-%d) a las $(date +%H:%M)"
26
27         else # sino logra reiniciar el servicio
28             mensaje="sempai el servicio $servicio no c pudo reiniar" # manda un mensaje donde notifica que no se pudo reinicar
29
30         fi # final de la condicion de reinicio
31
32         mensaje_telegram "$mensaje" # envia una notificacion si el servicio se (reinicio/o fallo)
33
34     else
35         echo " el servicio esta funcionando bien " #manda un mensaje a la terminal que los servicios estan bien (en el caso que ningun servicio este)
36     fi
37     echo "$mensaje"
38     echo "$mensaje" >> "$(dirname "$0")/servicios.logs"
39 done # final de bucle

GNU nano 7.2
1  #!/bin/bash
```

Indicara que se trabaja en bash

```
source "$(dirname "$0")/config.txt"
```

Se cargan las variables y la lista de servicios desde config.txt

```
mensaje_telegram() { #funcion para enviar las notificaciones
    local mensaje="$1"
```

```
curl -s -X POST "https://api.telegram.org/bot$TOKEN/sendMessage" \
    -d chat_id="$CHAT_ID" \
    -d text="$mensaje"
```

Curl: sirve para hacer una petición de envío de datos a la API

-s: Para un mejor orden y claridad

-X POST: sirve para el envío de datos al servicio

-d chat_id: será el ID del Bot que se encuentra en telegram

-d text: será el encargado de hacer la petición del mensaje tipo texto

SendMessage: indicara que quiere enviar un mensaje

```
for servicio in "${SERVICIOS[@]}"; do
    systemctl is-active --quiet $servicio

    if [ $? -ne 0 ]; then
        systemctl restart "$servicio"
        sleep 1
    fi
done
```

Se inicia el bucle con un for

- Si el servicio en la lista de servicios esta activo entonces pasará y no habrá necesidad de reiniciarlo

En el caso de que no enviara una condición de if

- El servicio será enlistado (ósea ira uno por uno) en nuestro archivo config.txt la cual, si detecta que esta inactivo o detenido, intentará reiniciarlo automáticamente y esperará un segundo para mayor organización

```
systemctl is-active --quiet "$servicio"
```

El servicio volverá hacer una revisión de estado del servicio, en caso del resultado enviará un mensaje

```
if [ $? -eq 0 ]; then
    mensaje="onni-chan, el servicio $servicio se detubo o se cayo, pero se reinicio correctamente, el reinicio se realizo el día $(date +%Y-%m-%d %H:%M:%S)"
else
    mensaje="sempai el servicio $servicio no c pudo reiniar"
fi
```

Aqui representa, que si el ultimo comando ejecutado es diferente a 0, significa que fallo, pero de lo contrario si es igual a 0, significa que funciono y enviara el mensaje correspondiente

```
mensaje_telegram "$mensaje"
```

Esta será una función que se le pasará los argumentos (los mensajes) para ser enviados

```

else
    echo " el servicio esta funcionando bien "
fi
    echo "$mensaje"
    echo "$mensaje" >> "$(dirname "$0")/servicios.logs"

```

En el caso que ningún servicio este detenido o caído, se enviara un mensaje a la terminal “que el servicio está funcionando bien” y esos mensajes se redirigirán a un archivo, logs especial para el control de registros

- config.txt: Contiene las variables TOKEN, CHAT_ID y la lista de servicios a monitorear.

```

GNU nano 7.2 config.txt
1 TOKEN=7862071230:AAGkPBjaXhVQqp6daSTrR0hrxmHny-1l17Q
2 CHAT_ID=-1002891538692
3 SERVICIOS=("ssh" "cron" "nginx")
4

```

- servicios.logs: Archivo de texto que almacena los eventos registrados.
- bucle.sh: Script adicional que ejecuta servicios.sh en un bucle infinito cada 10 segundos.

```

GNU nano 7.2 bucle.sh
1 #!/bin/bash
2
3 while true; do # inicio del bucle
4     sudo /home/irvin/GitHub/Proyecto-Prograservicios/servicios.sh # directorio de la ubicacion del script para el uso del bucle
5     sleep 10 # hace una breve espera de 10 seg, para poder ejecutar el script
6 done # fin del bucle
7

```

Pruebas

Para poder ejecutar el script de forma automática, primero debemos de utilizar el script de bucle.sh en segundo plano

```
irvin@debian:~/GitHub/Proyecto-Prograservicios$ nohup ./bucle.sh &_
```

Ssh

```
irvin@debian:~/GitHub/Proyecto-Prograservicios$ sudo systemctl stop ssh
```

```

onni-chan, el servicio ssh se detubo o se cayo, pero se reinicio
correctamente, el reinicio se realizo el dia 2025-06-23 en la
hora 00:50:00
00:50 AM

```

```

irvin@debian:~/GitHub/Proyecto-Prograservicios$ sudo systemctl status ssh
• ssh.service - OpenBSD Secure Shell server
  Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)
  Active: active (running) since Mon 2025-06-23 00:49:59 CST; 59s ago

```

Cron


```
irvin@debian:~/GitHub/Proyecto-Prograservicios$ sudo systemctl stop cron
```

onni-chan, el servicio cron se detubo o se cayo, pero se reinicio correctamente, el reinicio se realizo el dia 2025-06-23 en la hora 00:50:02

00:50 AM

```
irvin@debian:~/GitHub/Proyecto-Prograservicios$ sudo systemctl status cron
• cron.service - Regular background program processing daemon
  Loaded: loaded (/lib/systemd/system/cron.service; enabled; preset: enabled)
  Active: active (running) since Mon 2025-06-23 00:50:01 CST; 1min 24s ago
```

Nginx

```
irvin@debian:~/GitHub/Proyecto-Prograservicios$ sudo systemctl stop nginx
```

onni-chan, el servicio nginx se detubo o se cayo, pero se reinicio correctamente, el reinicio se realizo el dia 2025-06-23 en la hora 00:50:04

00:50 AM

```
irvin@debian:~/GitHub/Proyecto-Prograservicios$ sudo systemctl status nginx
• nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; preset: enabled)
  Active: active (running) since Mon 2025-06-23 00:50:03 CST; 1min 45s ago
```

Remoto.sh

Este script es el encargado de ejecutar scripts desde el equipo host a equipos en otras partes de la red de manera remota, facilitando y optimizando trabajo que podría ser más tedioso de manera presencial. Un SysAdmin normalmente gestiona varios servidores. Este script permite aplicar un mismo cambio, parche, monitoreo o configuración a todos al mismo tiempo, sin hacer clic en cada máquina ni repetir comandos uno por uno.

Script remoto.sh

```

1  #!/bin/bash
2
3  #Esrt script se encarga de ejecutar remotamente algun script implementado en este proyecto (u otro script)
4  #en diferentes equipos definidos en 'ips.txt'
5
6  # Verificamos que se reciban 3 argumentos: usuario, script y archivo con IPs/hosts
7  if [ "$#" -ne 3 ]; then
8      echo "Uso: $0 <usuario> <script.sh> <ips.txt>"
9      echo "Ejemplo: $0 jdope respaldo.sh ips.txt"
10     exit 1
11 fi
12
13 #Convertimos los parametros en variables para usarlos de forma mas facil dentro del script
14 Usuario="$1"
15 Script="$2"
16 Archivo="$3"
17
18 # Verificamos que el script exista en el equipo
19 if [ ! -f "$Script" ]; then
20     echo "Error: El archivo '$Script' no existe."
21     exit 1
22 fi
23
24 # Verificamos que el archivo 'ips.txt' exista
25 if [ ! -f "$Archivo" ]; then
26     echo "Error: El archivo '$Archivo' no existe o tiene un fallo"
27     exit 1
28 fi
29
30 #Confirmacion de que recibimos los dos archivos anteriores
31 echo "Archivos verificados. Ejecutando '$Script' en los hosts de '$Archivo'"
32
33 #(Si no existe) creamos el directorio donde se almacenan los reportes
34 mkdir -p Reportes
35
36 #Se analiza cada linea de 'ips.txt' en un bucle while donde se ejecuta en cada equipo
37 while IFS= read -r IP; do #IFS es un comando interno que analiza linea por linea un archivo (en este caso 'ips.txt')
38     echo "Analizando host: $IP"
39
40     #Si hay una linea vacia, la saltamos
41     if [ -z "$IP" ]; then
42         echo "Línea vacía. Saltando a la siguiente, se espera una IP"
43         continue
44     fi
45
46     #Creamos un nombre de log único por host + fecha (ej: 192.168.1.190_2025-06-23_01-45.log)
47     FechaLog=$(date +%Y-%m-%d_%H-%M)
48     Log="Reportes/${IP}_${FechaLog}.log"

```

```

49
50 #Escribimos el contenido del log
51 echo "📝 Log de ejecución remota para $IP" > "$Log"
52 echo "🕒 Fecha: $(date '+%Y-%m-%d %H:%M:%S')" >> "$Log"
53
54 #Copiamos el script al host remoto, en la carpeta temporal /tmp (ya que tmp borrara todo aquel contenido automaticamente en este directorio)
55 echo "Copiando $Script a $IP" | tee -a "$Log"
56 scp "$Script" "$Usuario@$IP:/tmp/" >> "$Log" 2>&1 #Redirecciona a la salida estandar de errores y asegura que tambien se ubique en el log
57 if [ $? -ne 0 ]; then
58     echo "Falló la copia a $IP" | tee -a "$Log"
59     continue
60 fi
61
62 # Ejecutamos el script remotamente
63 echo "Ejecutando script en $IP" | tee -a "$Log"
64 ssh "$Usuario@$IP" "bash /tmp/$(basename "$Script")" >> "$Log" 2>&1
65 if [ $? -eq 0 ]; then
66     echo "✅ Script ejecutado correctamente en $IP" | tee -a "$Log"
67 else
68     echo "⚠️ Hubo errores durante la ejecución en $IP" | tee -a "$Log"
69 fi
70
71 # Marcamos fin de log
72 echo "Finalizado $IP" | tee -a "$Log"
73 echo "-----" >> "$Log"
74
75 done < "$Archivo"

```

Ejecución y salida del script en terminal

```

root@basadope:/home/jdope/Proyecto_Adm.Servicios/Proyecto-Prograservicios# ./remoto.sh jdope prueba.sh ips.txt
Archivos verificados. Ejecutando 'prueba.sh' en los hosts de 'ips.txt'
Analizando host: 192.168.1.190
Copiando prueba.sh a 192.168.1.190
jdope@192.168.1.190's password:
Ejecutando script en 192.168.1.190
jdope@192.168.1.190's password:
[✓] Script ejecutado correctamente en 192.168.1.190
Finalizado 192.168.1.190
root@basadope:/home/jdope/Proyecto_Adm.Servicios/Proyecto-Prograservicios# █

```

En la salida podemos ver que pide doble contraseña, esto es porque usamos scp y ssh y exige contraseña para autenticar el acceso al sistema.

Resultado y verificación de uso

```

root@basadope:/tmp# ls -l
total 60
-rwxr-xr-x 1 jdope      jdope          39 jun 23 02:37 prueba.sh

```

La ilustración anterior es el host cliente 192.168.1.190 dentro del directorio /tmp donde se ubica la copia que hicimos vía scp.

Conclusión

Dirección de GitHub