

Script Automatización Redes

Tabla de contenido

Script Automatización Redes.....	1
Elementos y ficheros para su funcionamiento	1
Auto_redes_func5.py	2
Librerías:.....	2
Estructuras esenciales:	3
Func_lectura5.py:.....	4
Glosario de funciones y variables	8
Ejecución	8

Elementos y ficheros para su funcionamiento

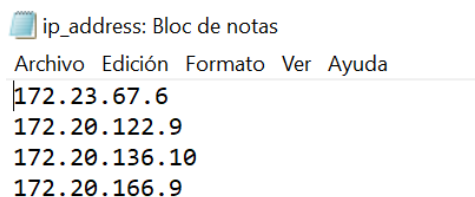
Este programa sirve para la automatización masiva de redes, para poder ejecutar de forma masiva comandos en elementos de red de forma rápida y eficiente de una sola ejecución. Este programa cuenta con dos archivos .py llamados:

auto_redes_func5.py: encargado de establecer la conexión ssh con los switches y contiene el bucle para la ejecución de comandos en cada switch.

func_lectura5.py: contiene las funciones de lectura de ficheros, lectura de logs, detección de pilas, puertos del switch...etc

Para el correcto funcionamiento, este script tiene 4 ficheros de texto:

- ip_address.txt: este fichero deberá contener las IPs de los dispositivos a los que se quiere acceder, una IP por línea.



ip_address: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
172.23.67.6
172.20.122.9
172.20.136.10
172.20.166.9
```

- autenticacionSwitch.txt: aquí se deberá escribir el usuario (primera línea del fichero) y contraseña (segunda línea) para poder acceder a los dispositivos

```
autenticacionSwitch: Bloc de notas
Archivo Edición Formato Ver Ayuda
xcristian
```

- comandos.txt: aquí se deberán incluir los comandos globales a ejecutar en el switch, ya sea en modo ejecución o en modo configuración.

```
comandos: Bloc de notas
Archivo Edición Formato Ver Ayuda
sh run | i vty
sh authentication session
sh vlan
conf t
!arp 172.20.146.210 0024.50de.1ec0 ARPA alias
do sh ver
exit
```

- comandos_int.txt: este fichero tendrá los comandos a ejecutar en modo configuración, a nivel de interfaz (previamente el script habrá detectado cuantos puertos tiene el dispositivo y si existe una pila)

```
comandos_int: Bloc de notas
Archivo Edición Formato Ver Ayuda
!switchport mode access
!no logging event link-status
!no snmp trap link-status
!authentication host-mode multi-auth
!authentication port-control auto
!authentication periodic
!authentication timer reauthenticate 14400
!mab
!dot1x pae authenticator
!spanning-tree portfast
exit
```

*en este fichero hay que escribir exit como último comando

Estos ficheros de textos deben estar ubicados en la misma carpeta que los archivos .py

Auto_redes_func5.py

Librerías: la librería esencial para el correcto funcionamiento y poder interactuar con el switch a través de ssh es paramiko. Otra librería esencial es la de logging, que permitirá la creación de logs de las salidas del CLI del switch. El resto de las librerías se necesitan para implementar alguna línea de código sin mayor importancia.

```
import paramiko
import time #paar tiempo de espera
import logging
import re
import func_lectura5 as fun
```

Destacar que la última línea es para importar las funciones del otro archivo .py, las llamadas a las funciones en este archivo deberán hacerse insertando fun. Al inicio de la línea.

Estructuras esenciales:

Bucle for de conexiones ssh

```
for i in range(0,num):

    direccion=str(ip[i])
    direccion=direccion.rstrip("\n")
    usuario= str(lineasg[0])
    usuario=usuario.rstrip("\n")
    contraseña= str(lineasg[1])
    contraseña=contraseña.rstrip("\n")

    con_ssh=paramiko.SSHClient()#crear un nuevo cliente ssh
    con_ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    con_ssh.connect(direccion, port=22, username=usuario, password=contraseña, look_for_keys=False, allow_agent=False)
    conexion=con_ssh.invoke_shell()

    num_comandos, comando = fun.lectura_comandos()
    salida2,nombre=fun.term_length(conexion)
```

Este bucle permite la conexión ssh con los dispositivos del fichero que contiene las IPs, habiendo leído previamente el usuario y contraseña

Bucle for de comandos

```
for a in range(0,num_comandos):

    palabra=comando[a].split()
    size=len(palabra)

    fun.comando_arp(palabra,comando,a,conexion, salida2,nombre)

    conexion.send(comando[a] + str("\n"))#mandar el comando
    time.sleep(1)#espera 1 segundos
    salida=conexion.recv(65535)#recibir los datos de ssalida
    salida=str(salida,'UTF-8')#convertir de bytes a string

    print(salida)
    logging.info(salida)
    with open((salida2), 'r', encoding="utf8") as f:
        lines = f.readlines()

    lines = [line for line in lines if line.strip()]
```

Este bucle ejecuta los comandos del fichero de comandos.txt uno por uno, y de forma secuencial. A continuación de este fragmento de código hay unos sentencias y condiciones “if” que dotan al script con la capacidad de entender si la ejecución de un comando ha finalizado. Por lo que, si la instrucción no ha terminado, no se ejecutará el siguiente comando.

*Cuando aparece la función `logging.info()` significa que lo que vaya entre paréntesis se guardará en el log, mientras que si pone `print`, solo se imprimirá por consola

Después se encuentra el siguiente fragmento de código, donde se llama a la función que se encarga de la detección de pilas, se eliminan los retornos de carro del log, se cierra la conexión ssh con el dispositivo y se eliminan las variables internas creadas de la sesión de logging.

```
fun.pila(conexion, salida2)
print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
print("ENTRANDO EN EL SIGUIENTE DISPOSITIVO...")

fun.quit_retorno_carro(salida2)

logging.shutdown()

con_ssh.close()

if logging.getLogger('').hasHandlers():
    logging.getLogger('').handlers.clear()

print("SCRIPT FINALIZADO, NO HAY MAS DISPOSITIVOS")
end = time.time()
print(format(end-start)+str(' segundos en ejecutar el script al completo'))
```

[Func_lectura5.py](#):

Este archivo de Python contiene varias funciones:

```
def lectura_ip():
    f=open("ip_address.txt")#abrir fichero
    lineas=f.readlines()#leer lineas
    num= len(lineas)#cuantas lineas hay
    ip=[]
    for i in range(0,num):#ir linea a linea y mostrar por pantalla
        ip.append(lineas[i])
    return num,ip
```

Esta función no tiene mucho misterio, lee cada línea del fichero de las direcciones IP y devuelve el número de direcciones que hay, y una lista con las propias IPs de cada switch

```
def lectura_usu():
    g=open("autenticacionSwitch.txt")
    lineasg=g.readlines()
    print("Usuarios es: ", lineasg[0])
    return lineasg

def lectura_comandos():
    h=open("comandos.txt")
    lineash=h.readlines()
    num= len(lineash)
    comando=[]
    for i in range(0,num):
        comando.append(lineash[i])
    return num,comando
```

Las siguientes funciones son prácticamente idénticas, la diferencia es que leen las credenciales (usuario y contraseña) y los comandos globales. Cada línea se guardará en una lista.

```
44
45 def loopback0_mac(conexion, salida2,nombre):
46
47     conexion.send("do sh ip int brief | i Loopback0\n")
48     time.sleep(1)
49     salida4=conexion.recv(65535)
50     salida4=str(salida4,'UTF-8')
51     logging.info(salida4)
52
53     conexion.send("do sh int vlan1\n")
54     time.sleep(1)
55     salida4=conexion.recv(65535)
56     salida4=str(salida4,'UTF-8')
57     logging.info(salida4)
58
59     with open((salida2), 'r', encoding="utf8") as f:
60         lines = f.readlines()
61
62     lines = [line for line in lines if line.strip()]
63
64     datos2=[]
65     for line in lines:
66         datos2.extend(line.split())
67
68     nombre=re.sub("\r","",nombre)
69     nombre=re.sub("\n","",nombre)
70     tam=len(datos2)
71     time.sleep(1)
72     for i in range(0,tam):
73         if datos2[i]=='i':
74             if datos2[i+1]=='Loopback0':
75                 if datos2[i+2]=='Loopback0':
76                     ip_loopback=datos2[i+3]
77                     print("La ip Loopback0 de %s es:%s"%(nombre,ip_loopback))
78
79             if datos2[i]=='address':
80                 if datos2[i+1]=='is':
81                     if datos2[i+3]=='(bia':
82                         mac=datos2[i+2]
83                         print("La direccion MAC de %s es:%s"%(nombre,mac))
84
85     return ip_loopback,mac
86
```

La siguiente función se utiliza para encontrar la IP de vlan1 y la dirección del Loopback0 para utilizar el comando arp...alias ARPA. Se ejecutan los comandos “do sh int vlan1” y “do sh ip int brief | i Loopback0” para obtener las respectivas direcciones. En las líneas 58 a 83, se separa las salidas en palabras y se hace un filtrado para encontrar las direcciones.

```
def term_length(conexion):
    salida=conexion.recv(65535)#recibir datos de socket tcp y udp (nº bytes a recibir)
    salida=str(salida,'UTF-8')#convertir de bytes a string
    print (salida)
    logging.info(salida)
    conexion.send("term Length 0\n")

    salida2=salida.rstrip("#")
    salida2=salida2.rstrip("\n")
    nombre=str(salida2)
    now =(datetime.today().strftime('%Y-%m-%d %Hh%Mmin'))
    now=str(now)
    # now=re.sub("\r","",now)
    # now=re.sub("\n","",now)

    now=re.sub("-", "_",now)
    now=re.sub(" ", "-",now)
    now=re.sub(":", "_",now)
    salida2=str(salida2)+str("-")+ str(now) +str("_Log.txt")
    salida2=re.sub("\r","",salida2)
    salida2=re.sub("\n","",salida2)

    if logging.getLogger('').hasHandlers():
        logging.getLogger('').handlers.clear()

    logging.basicConfig(format = '%(message)s',level = logging.INFO, filename = (salida2),filemode="w" )

    return salida2,nombre
```

Esta función se encarga de ejecutar el comando term length para que las salidas no se queden a medias, obtiene el nombre del switch y crea la sesión logging con la respectiva fecha y hora.

```
def comando_arp(palabra,comando,a, conexion, salida2, nombre):  
    if palabra[0]=='!arp':  
        if palabra[3]=='ARPA':  
            if palabra[4]=='alias':  
                ip_loopback,mac = loopback0_mac(conexion, salida2,nombre)  
                comando[a]='!arp %s %s ARPA alias'%(ip_loopback,mac))
```

Aquí se detecta si existe el comando arp...ARPA alias en el fichero de comandos. Cabe destacar que la ip de vlan1 y loopback0 que se escriben en el fichero de comandos es indiferente, el script lo sustituirá por las direcciones correctas.

```
def pila(conexion,salida2):  
    # logging.basicConfig(format = '%(message)s',level = logging.INFO, filename = (salida2),filemode="w" )  
  
    conexion.send("sh ver | i Model [Nn]umber\n")  
    time.sleep(1)  
    salida4=conexion.recv(65535)  
    salida4=str(salida4,'UTF-8')  
    logging.info(salida4)  
  
    with open((salida2), 'r', encoding="utf8") as f:  
        lines = f.readlines()  
  
    lines = [line for line in lines if line.strip()]
```

Esta función de detección de pila es la más extensa en cuanto a líneas de código, es la encargada de detectar cuántos switches existen, el número de puertos y el tipo de estos (fast, giga..etc). El número de puertos y la cantidad de switches se detecta con el comando "sh ver | i Model [Nn]umber". El resto de la función es simplemente tratamiento de cadenas y listas.

Destacar las siguientes variables:

- puerto: es una lista en la que cada elemento indica el número de puertos de cada switch, es decir, el primer elemento es el nº de puertos del primer switch y así sucesivamente. Ejemplo: [48,48,24,24]
- gg: es el número total de switches, independientemente de los puertos que tengan
- numero_gg: es una lista con los números de cada switch. Ejemplo: [1,2,3...]

```

def detec_puertos(gg, numero_gg, puerto, conexion, tipo_puerto, rango, n48, n24):
    numj, comandoj = lectura_comandos_int()
    tam1 = len(comandoj)

    if tam1 == 0:
        print('El fichero de comandos para interfaces está vacío')
        return
    conexion.send("conf t\n")
    time.sleep(1)
    salida4 = conexion.recv(65535)
    salida4 = str(salida4, 'UTF-8')
    logging.info(salida4)
    print(salida4)

    for i in range(0, gg):
        if (n48 > 1 or n24 > 1):
            comando = str('int range ') + str(tipo_puerto) + str(numero_gg[i]) + str(rango) + str(puerto[i]) + str('\n')
        else:
            comando = str('int range ') + str(tipo_puerto) + str(rango) + str(puerto[i]) + str('\n')
        conexion.send(comando)
        time.sleep(1)
        salida4 = conexion.recv(65535)
        salida4 = str(salida4, 'UTF-8')
        logging.info(salida4)
        print(salida4)
        #for de lectura de comandos_int
        ejecucion_comandos_int(numj, comandoj, conexion)

    conexion.send("exit\n")
    time.sleep(1)
    salida4 = conexion.recv(65535)
    salida4 = str(salida4, 'UTF-8')
    logging.info(salida4)

```

En esta función de la imagen anterior, es la encargada de ejecutar el comando para la selección del rango de interfaces de los switches de cada pila. Las variables n48 y n24 indican la cantidad de switches que hay con puertos 48 y 24 respectivamente que hay en pila.

En el caso de que haya pila, el comando que se ejecutará será del estilo int range g1/0/1-24. En caso contrario, una instrucción de este estilo, int range g0/1-24. Esta diferenciación se encuentra en el if y else que definen la variable comando.

```

def ejecucion_comandos_int(numj, comandoj, conexion):

    for a in range(0, numj):
        conexion.send(comandoj[a] + str("\n")) #mandar el comando
        time.sleep(0.5) #espera 1 segundos
        salida = conexion.recv(65535) #recibir los datos de salida
        salida = str(salida, 'UTF-8') #convertir de bytes a string

        print(salida)
        logging.info(salida)

def lectura_comandos_int():
    j = open("comandos_int.txt")
    lineasj = j.readlines()
    numj = len(lineasj)
    comandoj = []
    for i in range(0, numj):
        comandoj.append(lineasj[i])
    return numj, comandoj

def quit_retorno_carro(salida2):
    with open((salida2), 'r', encoding="utf8") as f:
        lines = f.readlines()
        lines = [line for line in lines if line.strip()]
        datos = []
        for line in lines:
            datos.append(line.strip('\n'))
    with open((salida2), 'w', encoding="utf8") as f:
        f.writelines(lines)

```

En la imagen anterior tenemos la función de ejecución_comandos_int, que básicamente es un bucle para ejecutar los comandos del fichero comandos_int.txt.

Lectura_comandos_int, bucle encargado de guardar en una lista los comandos dedicados a las interfaces. La función quit_retorno_carro de primeras es una función bastante “fea”, abre el log y guarda cada línea de este en una lista, para cada elemento de la lista elimina el retorno “\n”, para después abrir el fichero en modo escritura “w” y sobrescribe el txt con las líneas sin el retorno de carro.

Glosario de funciones y variables

A continuación se exponen variables y funciones que pueden dar lugar a confusión

- ❖ logging.info(x): función para guardar x en el log creado
- ❖ .readlines(): guardar en cada elemento de una lista las líneas de un objeto
- ❖ time.sleep(x): esperar x segundos en la ejecución del programa
- ❖ conexión.send: enviar un comando al CLI del switch (la variable conexión se ha referenciado como conexión ssh)
- ❖ conexión.recv: recibir la salida del CLI
- ❖ puerto: es una lista en la que cada elemento indica el número de puertos de cada switch, es decir, el primer elemento es el nº de puertos del primer switch y así sucesivamente. Ejemplo: [48,48,24,24]
- ❖ gg: es el número total de switches, independientemente de los puertos que tengan
- ❖ numero_gg: es una lista con los números de cada switch. Ejemplo: [1,2,3...]
- ❖ salida2: variable de tipo cadena que guarda el nombre de switch+fecha+hora de cuando se estableció la conexión ssh
- ❖ n48: cantidad de switches que hay en pila con 48 puertos
- ❖ n24: cantidad de switches que hay en pila con 24 puertos

Ejecución

Para ejecutar el script es bastante sencillo, bastará con abrir en el editor Spyder los archivos .py, y ejecutar el archivo auto_redes_func5.py

También se puede ejecutar desde cmd ejecutando el comando “python auto_redes_func5.py” en la carpeta donde esté ubicado el archivo