Automatización de requerimientos y configuraciones en equipos de red del área de conectividad y seguridad perimetral de la empresa ARUS S.A. para mejora en eficiencia de tiempo mediante programación de scripts.

Autor:

Julián Andrés Zapata Díaz

Informe de práctica como requisito para optar al título de:
Ingeniero de Telecomunicaciones.

Asesor interno:
Jaime Alberto Vergara Tejada

Asesor externo:
Jorge Alberto Vallejo Vélez

Universidad de Antioquia
Facultad de ingeniería, departamento de ingeniería Electrónica y Telecomunicaciones.
Medellín, Colombia
2022.

Cita	(Zapata Diaz, 2022)
Referencia **Estilo APA 7 (2020)**	Zapata Díaz, J. A. (2022). *Automatización de requerimientos y configuraciones en equipos de red del área de conectividad y seguridad perimetral de la empresa ARUS S.A. para mejora en eficiencia de tiempo mediante programación de scripts.* Práctica profesional. Universidad de Antioquia, Medellín.

Agradecimientos

Asesor interno profesor de la facultad de ingeniería: Jaime Alberto Vergara.

Asesores externos: Jorge Alberto Vallejo & Andres Felipe Ortiz.

Contenido

1 Resumen

Dentro de las tareas que se realizan en el área de conectividad y seguridad perimetral de la compañía ARUS S.A se encuentra la implementación y configuración de equipos de red tales como switches, routers, controladoras inalámbricas, puntos de acceso inalámbricos, firewalls, etc. De diferentes marcas y donde en la mayoría de dispositivos la administración y/o configuración se hace a través de la línea de comandos o CLI (Command-Line Interface) mediante el protocolo de acceso remoto SSH (Secure SHell) donde los comandos (sentencias gramaticales del lenguaje de fabricante) usados son bien conocidos, de uso constante y repetitivo. En consecuencia, se procedió a determinar dichas tareas y/o requerimientos y se determinó la viabilidad y consiguiente automatización mediante una herramienta que permita al ingeniero mejorar los tiempos de respuesta, liberar más tiempo para otras actividades, disminuir el factor de error humano, disminuir complejidad de operaciones, entre otras ventajas, manteniendo los estándares de seguridad en la conexión con los dispositivos.

Para la ejecución del proyecto se realizó la búsqueda de una herramienta de fácil implementación y de libre uso (sin pago por licencia) que permitiera la automatización de las tareas cumpliendo con estándar de conexión segura determinado por el grupo de trabajo en la compañía, encontrándose la librería del lenguaje de programación Python ***Netmiko.***

Se procedió con la recopilación de información tanto del uso de la librería como de los comandos/tareas a automatizar con los cuales se construyó un script que establece conexiones seguras con los equipos de red y ejecuta comandos y grupos de comandos (rutinas) en los mismos, adicionalmente para lograr una interacción fácil e intuitiva con el usuario se creó una interfaz gráfica con la librería TKINTER.

El desarrollo del script resultado se realizó en el entorno de pruebas GNS3 con imágenes virtuales de equipos de red de diferentes proveedores y máquina virtual con sistema operativo Linux. El script resultante es funcional y permite la automatización de tareas como backups (copias de respaldo de configuración) de equipos y ejecución de plantillas de comandos preestablecidas y personalizables.

2 Introducción

El equipo de conectividad y seguridad perimetral de la compañía ARUS S.A es responsable por la administración de la infraestructura de red de diversos clientes, es decir monitoreo, configuración, solución de problemas y ejecución de mejores prácticas bien sea por recomendación de fabricante o necesidad particular de cliente, las cuales permitan un funcionamiento correcto de la red de datos y se establezca un perímetro seguro para los usuarios y activos de información.

Las motivaciones que se encuentran para la realización del proyecto son 2 en particular:

- Muchos requerimientos de configuración de los diferentes clientes son similares y repetitivos lo que conlleva a realizar la misma tarea reiteradas veces.

- Existen clientes con infraestructuras de red muy grandes es decir con muchos dispositivos y las configuraciones que se deben desplegar de manera masiva suponen una inversión considerable de tiempo y riesgo por error humano.

Dichos antecedentes exponen la necesidad de plantear nuevas estrategias que permitan la solución eficaz a ciertas tareas/requerimientos donde con un previo análisis a cada una de estas en particular se determine su viabilidad hacia la automatización.

Se identifican como dificultades que los diferentes clientes tienen equipos de diversos fabricantes cuya interfaz de administración y lenguaje de comandos varía según el fabricante, se deben mantener estándares de seguridad donde la herramienta a usar no constituya una vulnerabilidad y la transmisión de la información utilice protocolos seguros, además de una interacción cómoda entre el usuario y la herramienta ésta debe ajustarse o tener la capacidad de personalizarse fácilmente por el usuario con rutinas de acuerdo a la necesidad particular de cada cliente.

La propuesta constituye la exploración, elaboración e implementación en ambiente de pruebas de una herramienta que permita la automatización de rutinas de configuración, se establece el alcance hasta su implementación en ambiente de pruebas por la limitación en tiempo para la entrega de proyecto y dado que se realizan configuraciones sobre equipos, estas pruebas no deben realizarse sobre equipos productivos de clientes.

Teniendo en cuenta lo anterior y las recomendaciones por parte de los asesores para la realización del proyecto se definió Python para crear los scripts de automatización mediante el uso de una librería en particular, que permite el fácil y seguro acceso a dispositivos de red mediante el protocolo SSH (Secure Shell), llamada Netmiko (Samoylenko, 2022).

3 Objetivos

3.1 Objetivo general

Desarrollar e implementar en entorno de pruebas una aplicación para automatizar rutinas de requerimientos por parte del clientes y configuraciones de equipos de red como switchs, routers, puntos de acceso inalámbrico y firewalls mediante Implementación de scripts que permitan una solución más eficaz y manejable que disminuya el tiempo y complejidad de ejecución de dichas tareas.

3.2 Objetivos específicos

1. Seleccionar las tareas y requerimientos a automatizar según las necesidades del área de conectividad y seguridad perimetral, definir los comandos a utilizar según el fabricante mediante la determinación de los equipos de red (marca, modelo y versión de firmware) donde se llevarán a cabo dichas automatizaciones.

2. Construir un entorno de pruebas donde se simulan equipos de red necesarios para la automatización de tareas y requerimientos seleccionados que permita crear e implementar scripts de automatización.

3. Realizar un protocolo de evaluación con el fin de corregir errores y revisar la viabilidad de implementación de la solución en un entorno real en la compañía.

4 Marco Teórico

La automatización en configuraciones de equipos de red son una tendencia global impulsada por las redes definidas por software (SDN) que mediante la automatización por scripting aminoran tiempos de configuración, costos operacionales y reducen errores ocasionados por humanos como se menciona en (Mihăilă, Bălan, Curpen & Sandu, 2017), especialmente en redes medianas y grandes donde hay varios equipos de red que a su vez corresponden a diversos fabricantes. Lo anterior, dificulta la administración de la red puesto que cada fabricante utiliza un lenguaje de configuración diferente, aunque en algunos casos parecidos unos con otros. Grandes fabricantes de dispositivos como Cisco han promovido la configuración de redes por software mediante lenguajes de programación abiertos y de amplia difusión como es el caso de Python pero existen diversas herramientas que incluyen características diferentes, algunas con versiones de licencia paga y otros de uso libre sin ningún costo como las que se mencionan en (Parker, 2019), características como por ejemplo analizadores de tráfico de red, programación de copias de seguridad o configuraciones de dispositivos, inventario automático de dispositivos de red, etc. También destaca que algunas soluciones tienen interfaz gráfica para la presentación de datos. Se pretende entonces la elección de una solución y adicionar cualidades que se encuentran en otras herramientas.

A continuación, se presentan los conceptos claves para el entendimiento del proyecto:

Paramiko: Es una librería que implementa el protocolo SSH, comúnmente utilizado para administrar sistemas UNIX de forma remota (Zadka, 2019). Según el sitio oficial paramiko 'proporciona funcionalidad de cliente y servidor. Si bien aprovecha una extensión Python C para criptografía de bajo nivel, Paramiko en sí es una interfaz pura de Python en torno a los conceptos de redes SSH'. (Forcier, n.d)

Netmiko: Esta librería ayuda a usuarios a simplificar funciones de protocolos de comunicación con dispositivos de red. Se basa en la librería Paramiko para los niveles bajos de conectividad SSH, pero provee una mejor abstracción en la comunicación con una variedad de modelos de dispositivos de red. Netmiko soporta un amplio rango de dispositivos y tienen una secuencia de comandos mucho más corta que la guía de código paramiko. (Slattery, 2020)

Scripting: Es un lenguaje de programación el cual tiene una diferencia primordial con respecto a otros lenguajes como C++ y es que un programa escrito en lenguaje scripting se interpreta en lugar de ser compilado. En los lenguajes interpretados, el texto que contiene los comandos es leído por un intérprete que realiza la conversión a código de máquina en sí mismo, mientras ejecuta el script. El texto aquí sigue siendo legible por humanos y no es necesario volver a compilarlo si se realiza un cambio. (Andress & Linn, 2017)

SSH (Secure Shell): Es un protocolo de conexión remota con métodos de autenticación y canal cifrado para administración y configuración de equipos de red como switch, routers, servidores, firewalls, etc. Se inventó originalmente como una alternativa segura al comando "telnet", pero pronto se convirtió en la herramienta de administración remota de facto. Incluso los sistemas que utilizan agentes personalizados para administrar una flota de servidores, como Salt, a menudo se iniciarán con SSH para instalar los agentes personalizados (Zadka, 2019), Existen 2 versiones de este protocolo, por mejor práctica en seguridad siempre que sea posible se debe implementar la versión 2 ya que esta 'utiliza un algoritmo de cifrado de seguridad mejorado'. (Cisco, 2020)

5 Metodología

A continuación, se describen las actividades que permitieron la consecución de los objetivos.

Objetivo № 1: Seleccionar las tareas y requerimientos a automatizar según las necesidades del área de conectividad y seguridad perimetral, definir los comandos a utilizar según el fabricante mediante la determinación de los equipos de red (marca, modelo y versión de firmware) donde se llevarán a cabo dichas automatizaciones.

Actividades definidas para la consecución del objetivo:

1. Realizar un inventario de los dispositivos donde se llevarán a cabo las automatizaciones, precisando su marca, modelo y versión.
2. Selección preliminar de tareas o rutinas candidatas a posible automatización mediante socialización con el equipo de trabajo.
3. Recolectar comandos (instrucciones) propios de cada fabricante según los dispositivos definidos en la anterior actividad.
4. Evaluar los requerimientos de las rutinas a automatizar y realizar una selección final.

Ejecución de actividades:

Para la definición de dispositivos a usar y los comandos o rutinas de mayor uso se realizó la solicitud en varias ocasiones a miembros del equipo de conectividad para que compartieran desde su experiencia los equipos y comandos donde tienen mayor interacción, encontrándose 4 fabricantes principales mencionados en orden de mayor uso a menor: Cisco, Fortinet, Aruba, 3COM (Comware) y UniFi. De la lista de equipos se seleccionan los dos (2) fabricantes principales para la recolección de comandos, los cuales se recopilan de la misma manera consultando al equipo de trabajo y de experiencia propia.

La recolección de rutinas y comandos destaca los siguientes requerimientos como los de mayor solicitud:

- Automatización de backups
- Creación, configuración y cambio de VLANs en puertos de acceso y troncales.
- Configuración de protocolo de árbol de expansión (STP)
- Configuración de políticas de navegación en equipo de perímetro firewall
- Configuración de port-security
- Configuración de timing a políticas de navegación
- Implementación de SD-WAN
- Configuración de protocolo 802.1X

- Configuración de traffic-shaping
- Configuración de perfiles de seguridad (Web filter, App control)

Para ver los comandos recolectados para algunas rutinas ver (Anexo 9.1.)

Con la información obtenida se procede con la elaboración del entorno de pruebas que permita la implementación del script.

Objetivo № 2: Construir entorno de pruebas donde se simulan equipos de red necesarios para la automatización de tareas y requerimientos seleccionados que permita crear e implementar scripts de automatización.

Actividades definidas para la consecución del objetivo:

1. Esquematizar el script con el número de casos que deben crearse según el dispositivo a automatizar.
2. Construir un entorno de pruebas con imágenes virtualizadas de los dispositivos de red necesarios.
3. Construcción de script con cada rutina y de dispositivo según lo establecido en las actividades anteriores en el entorno de pruebas.

Ejecución de actividades:

Los equipos donde se realizaron las rutinas de automatización son: Router cisco imagen IOS C7200-ADVENTERPRISEK9-M, Versión 15.0(1)M, Switch cisco imagen IOS vios_l2-ADVENTERPRISEK9-M Experimental Version 15.2 porque estos equipos permiten una gran cantidad de comandos de configuración y equipo Fortigate FGT_VM64_KVM-v6-build1828 con versión de IOS 6.4.5. recomendación de plataforma GNS3 por estabilidad y compatibilidad.

La topología resultante realizada en el ambiente de pruebas GNS3 es la siguiente (imagen 1).

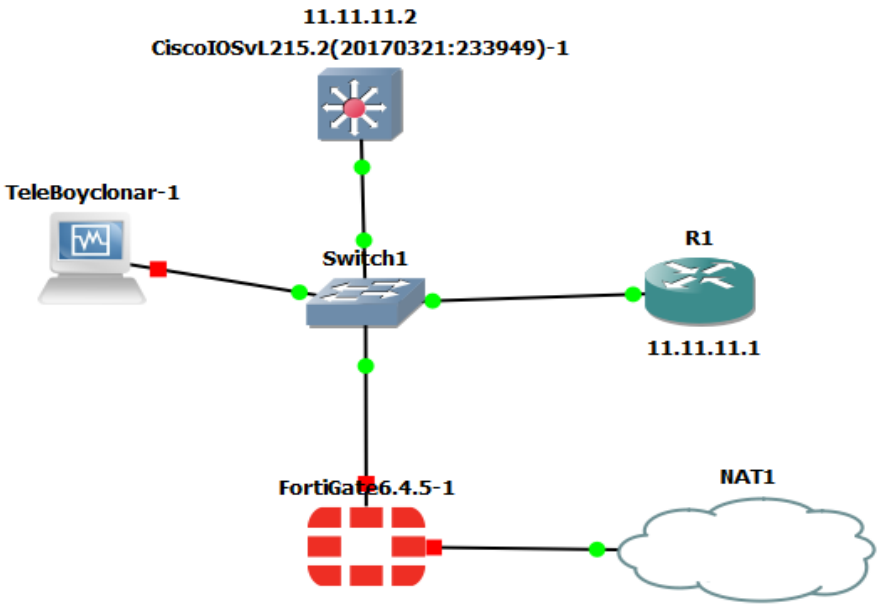

Imagen 1. Topología ambiente de pruebas GNS3.

En el dispositivo de nombre "TeleBoyclonar-1" con sistema operativo Linux se instala lenguaje de programación Python con librerías necesarias para la elaboración del aplicativo.

El script resultado es explicado en la sección de resultados y análisis.

Objetivo № 3: Realizar un protocolo de pruebas con el fin de corregir errores y revisar la viabilidad de implementación de la solución en un entorno real en la compañía.

Actividades definidas para la consecución del objetivo:

1. Solucionar errores en script y determinar métricas que permitan realizar un protocolo de evaluación.
2. Proponer viabilidad de las automatizaciones para su implementación como solución en equipos físicos no simulados.

Ejecución de actividades:

Depuración de objetos, clases y métodos redundantes, selección de construcción de script con arquitectura de capas (presentación y datos), para lograr una comprensión clara del flujo de programación.

El aplicativo resultante es evaluado en el ambiente de pruebas (imagen 1) realizando ejecución de rutinas de comandos en diferentes dispositivos, carga y ejecución de rutinas preestablecidas y personalizadas.

6 Resultados y análisis

Como requisito para lograr la comunicación entre el aplicativo y los equipos de red estos deben tener previa configuración del servicio SSH y conocer las credenciales de acceso, con esta configuración aplicada es posible realizar cualquier configuración desde el aplicativo.

Para lograr una interacción fácil e intuitiva se realiza una interfaz gráfica con el uso de la librería TKinter (Graphical User Interfaces with Tk — Python 3.10.4 documentation, z.d) de Python, dicha interfaz cuenta con 3 vistas o secciones (Imágenes 2-4). Donde la primera vista corresponde a la consulta de credenciales de acceso al dispositivo y la selección del sistema operativo, el alcance del aplicativo logra conectarse a un dispositivo a la vez, sin embargo, es posible y se constituye un atributo deseado que se logre la conexión a múltiples dispositivos de forma simultánea o contigua. Una vez realizado el correcto diligenciamiento de las credenciales de dispositivo se selecciona el sistema operativo (IOS), el aplicativo tiene la capacidad de informar cuando se produce un error en el establecimiento de la conexión mediante mensajes en pantalla (Imagen 5). Finalmente se establece sesión con el dispositivo y las pestañas o vistas de comandos son habilitadas.

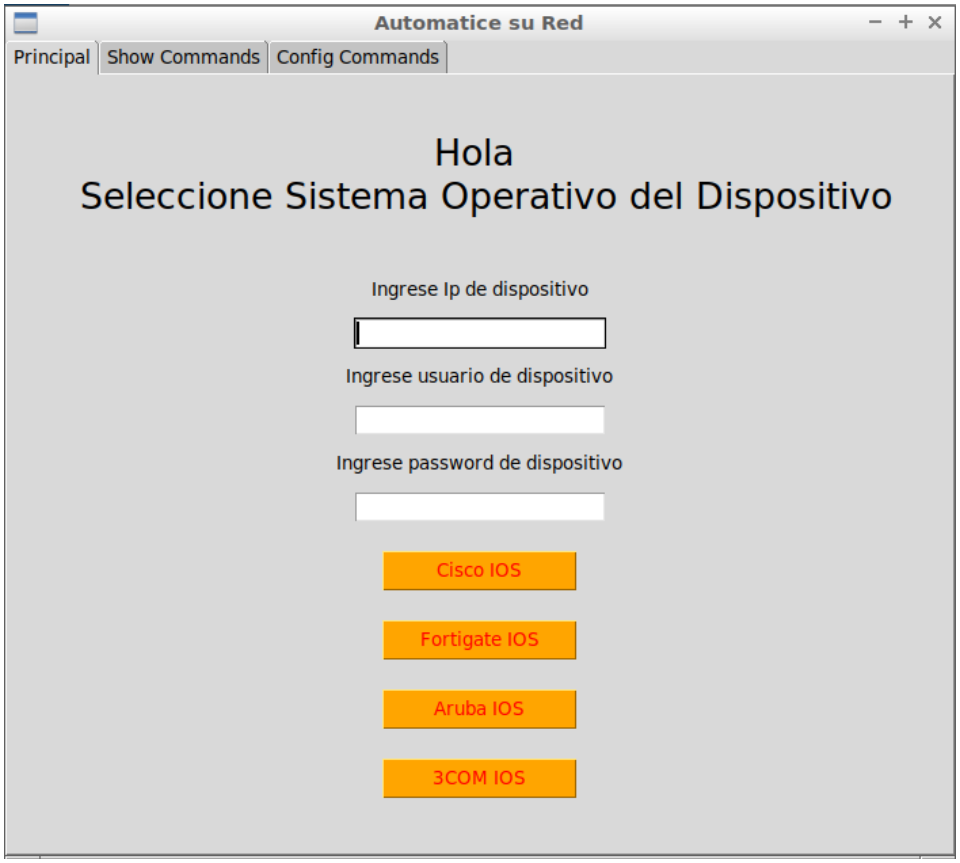

Imagen 2. Vista principal interfaz gráfica de aplicativo. configuraciones para iniciar sesión.

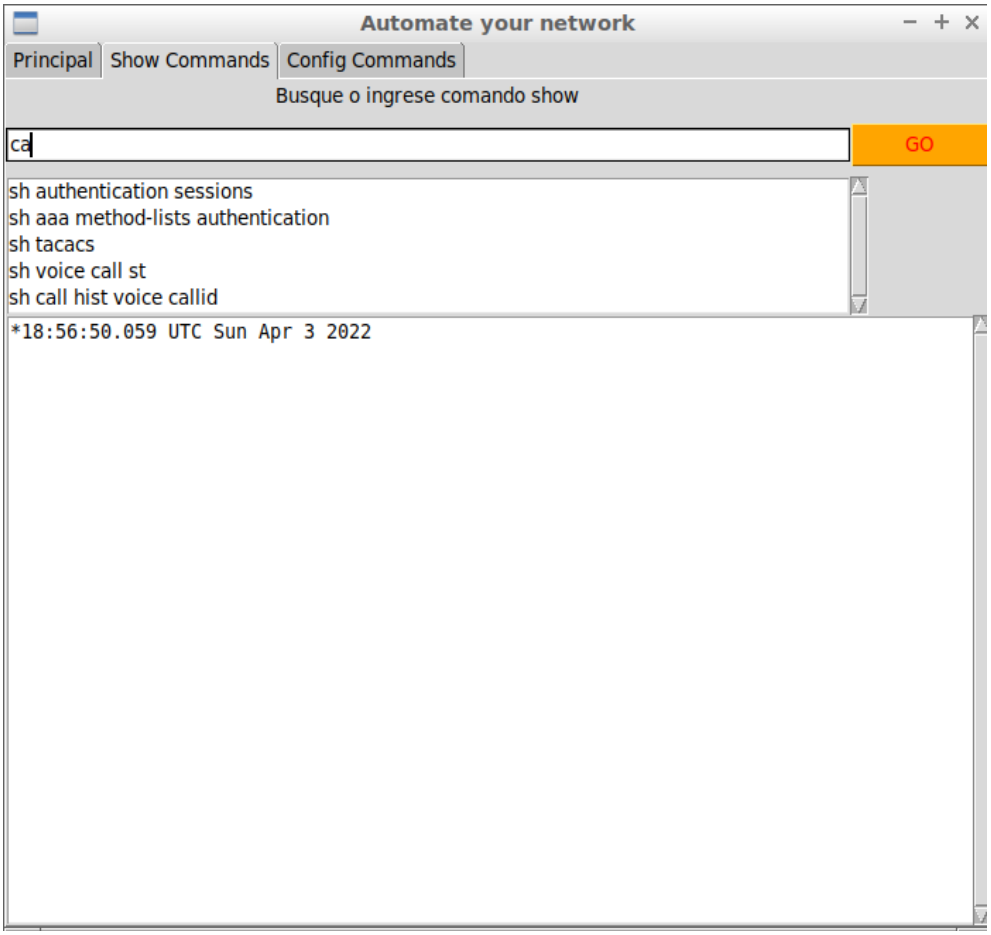

Imagen 3. Vista segunda ventana interfaz gráfica de aplicativo. Sección de comandos SHOW.

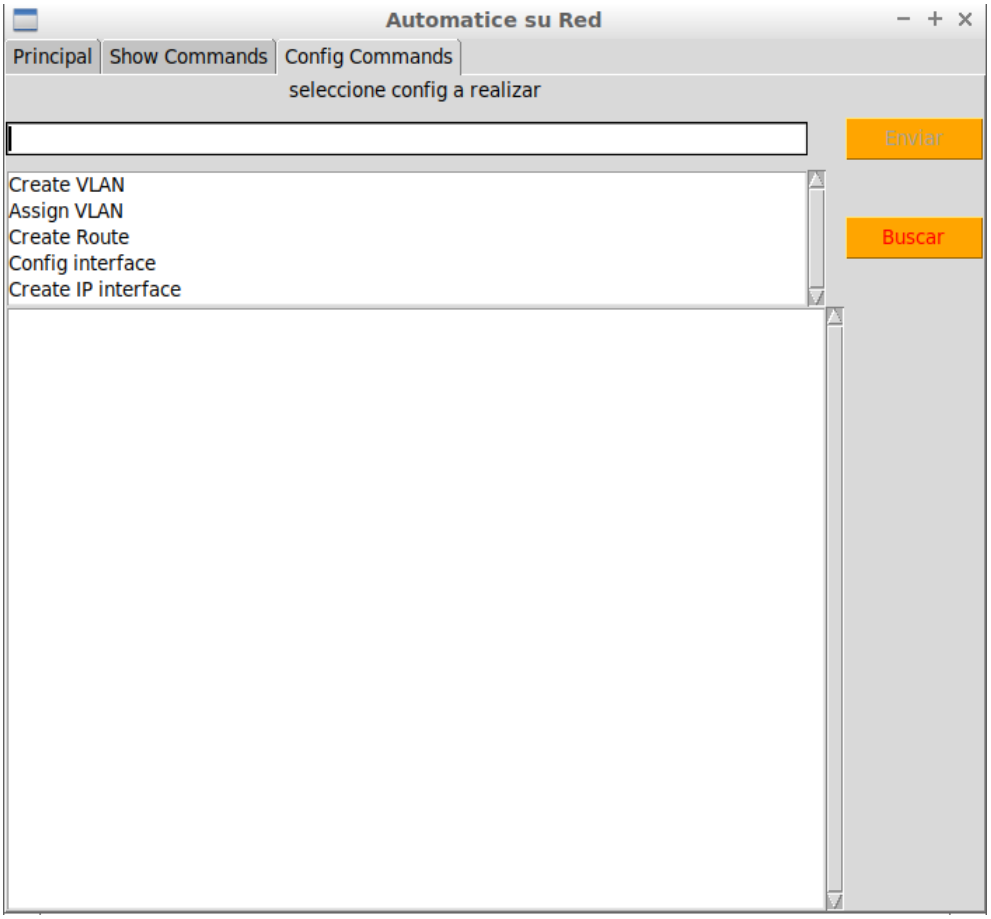

Imagen 4. Vista tercera ventana interfaz gráfica de aplicativo. Sección de comandos CONFIG.

Para la vista # 2 (Imagen 3) correspondiente a la sección de comandos SHOW se realiza la configuración de un buscador, un cuadro de despliegue de lista, tablero principal de visualización de resultados y botón de envío de comando. La lista de comandos show es un archivo en formato TXT independiente donde se encuentran todos los comandos tipo SHOW el cual puede ser reorganizado y personalizado por el usuario agregando comandos nuevos. El envío de comandos para esta sección se realiza uno a la vez, borrando el tablero de visualización para cada envío, es decir, solo se observa en pantalla el último comando SHOW enviado. Se destaca que para los comandos donde se debe esperar un tiempo para recibir un retorno como por ejemplo los comandos logging y debug muy usados para realizar troubleshootings no son compatibles con la estrategia de programación del script.

La visualización de comandos SHOW funciona de manera correcta y en comandos que superan el espacio de la pantalla como por ejemplo el comando "show run" el usuario puede ver la totalidad de la respuesta desplazándose por el tablero de forma vertical sin necesidad de uso de teclas adicionales.

La vista # 3 o vista de comandos de configuración (Imagen 4) cuenta con idénticas características que la vista # 2, con excepción de un nuevo botón de búsqueda. Para esta sección se planteó la estrategia de cambiar la interfaz gráfica de acuerdo a la configuración requerida realizando el despliegue de widgets adicionales según la cantidad de "variables" necesarias para la configuración, cuya funcionalidad fue probada, más dicha estrategia no corresponde a una solución eficaz por dos factores principales: 1) El número de variables depende de la necesidad específica del usuario 2) el número de variables puede ser muy grande lo cual expone la inviabilidad de la creación de una interfaz para cada configuración. por lo cual se establece una nueva estrategia al dotar al aplicativo con la capacidad de leer y ejecutar plantillas de rutinas personalizadas por el usuario mediante el botón "Buscar" (Ver imagen 4).

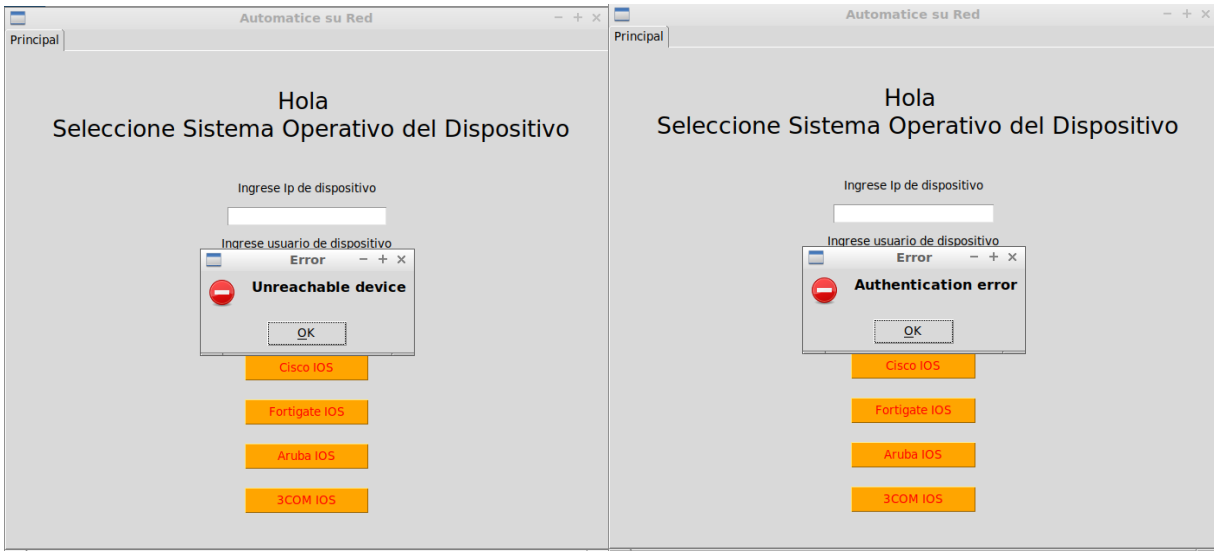

Imagen 5. Vista errores en el establecimiento de sesión SSH.

Para dar claridad al funcionamiento del código en Python (Ver Anexos) remítase a los diagramas de operación (Diagrama 1-3). el script está compuesto por 2 clases: "DEVICEIOS" la cual tiene los métodos o funciones que ejecuta Netmiko para la conexión, envío y retorno de respuesta de los comandos, por otro lado, la clase "UI" soporta todos los métodos de la interfaz gráfica, donde ambas clases se comunican mediante los objetos "APP" objeto de la clase "UI" y el objeto "DEVICE_X" de la clase "DEVICEIOS".

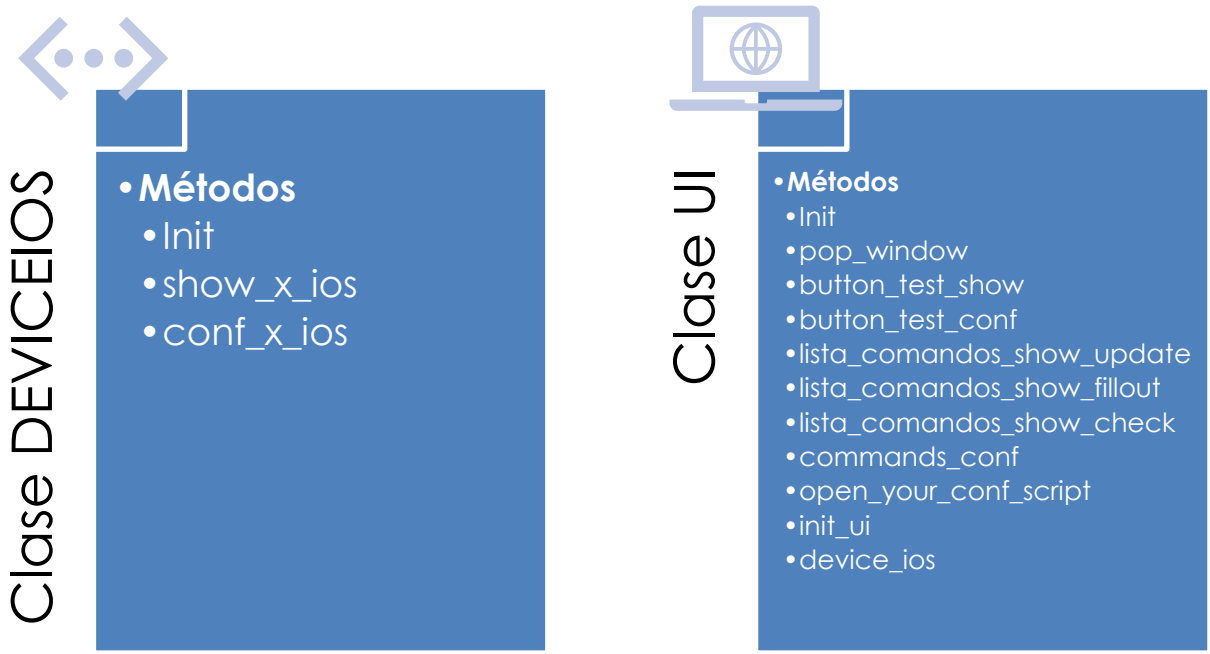

Diagrama 1. Diagrama de clases y métodos del script

Explicación métodos:

- Clase DEVICEIOS

Método "Init"	Método "show x ios"	Método "conf x ios"
•Recibe las credenciales del dispositivo y establece la conexión. •Retorna conexión exitosa o mensaje de error.	•Recibe comando *show* seleccionado por usuario y lo ejecuta en dispositivo de red. •Retorna respuesta de la ejecución del comando.	•Recibe set o conjunto de comandos *configuration* seleccionados por el usuario y los ejecuta en el dispositivo de red. •Retorna evidencia de la ejecución de cada comando y la correspondiente respuesta del dispositivo.

Diagrama 2. Operación de métodos de la clase DEVICEIOS.

- Clase UI

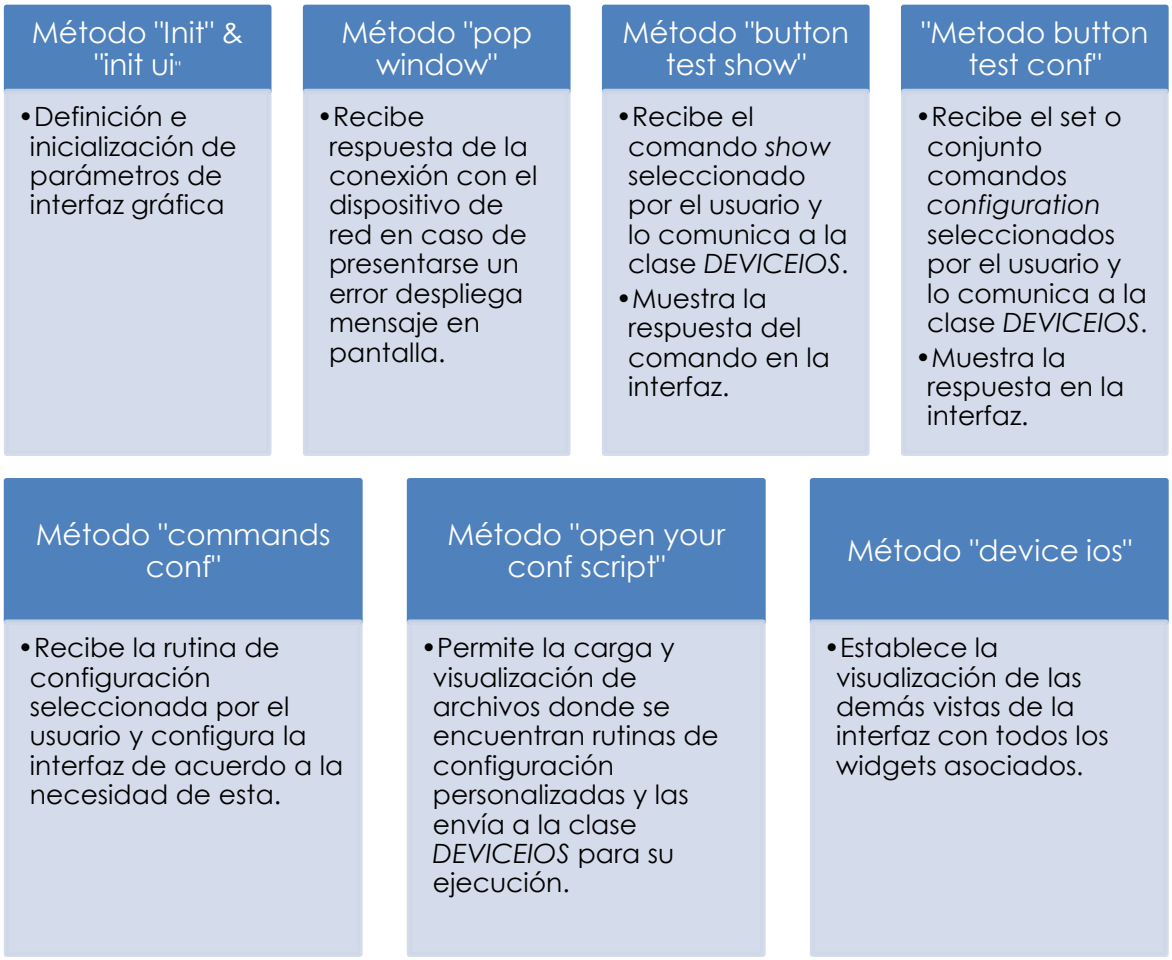

Diagrama 3. Operación de métodos de la clase UI.

Los métodos no mencionados son necesarios para el funcionamiento del widget de búsqueda rápida.

Verificación de funcionamiento en sistema operativo (Windows) correcto, se encuentra necesario cambiar las rutas de carga de los archivos TXT, una vez realizado este cambio el aplicativo funciona sin ningún problema.

7 Conclusiones

El aplicativo permite la ejecución de comandos o rutinas de comandos disminuyendo el factor de error humano, y se destaca su uso para el despliegue de configuraciones de forma masiva en infraestructuras donde varios equipos de red comparten las mismas credenciales de acceso.

No fue configurada la funcionalidad de realizar varias conexiones a varios equipos de forma secuencial sin embargo su elaboración no es compleja y puede lograrse mediante ciclos lógicos como el ciclo FOR.

Para la automatización de las tareas/rutinas de configuración se realiza la selección de rutinas cortas ya que según la rutina la interfaz gráfica debe cambiar para solicitar al usuario todas las variables necesarias, por lo cual se evidencia que cada rutina requiere un número de variables como nombres, ID, direcciones IP, entre otras que deben ser establecidos por el usuario, dicho número de variables depende de la configuración a realizar y una interfaz gráfica cambiante de acuerdo a la configuración no es viable puesto que el número de variables no es fijo sino que obedece a una necesidad particular por tanto se opta por que el usuario cargue al aplicativo su propia plantilla de configuración lo cual alienta al usuario a establecer y/o formar una biblioteca de plantillas de configuración.

8 Referencias Bibliográficas

Andress, J., & Linn, R. (2017). Introduction to command shell scripting. Coding for Penetration Testers, 7–42. doi:10.1016/b978-0-12-805472-7.00001-2

Byers, K. (z.d.). GitHub - ktbyers/netmiko: Multi-vendor library to simplify paramiko SSH connections to network devices. GitHub. Retrieved 4 April 2022, from https://github.com/ktbyers/netmiko

Cisco. (2020, December 24). Configuring Secure Shell on Routers and Switches Running Cisco IOS. https://www.cisco.com/c/en/us/support/docs/security-vpn/secure-shell-ssh/4145-ssh.html

Forcier, J. (n.d.). Welcome to Paramiko! — Paramiko documentation. Http://Www.Paramiko.Org/.

Graphical User Interfaces with Tk — Python 3.10.4 documentation. (z.d.). docs.python.org. Retrieved 4 April 2022, from https://docs.python.org/3/library/tk.html

Mihăilă, P., Bălan, T., Curpen, R., & Sandu, F. (2017). Network Automation and Abstraction using Python Programming Methods. MACRo 2015, 2(1), 95-103.

Parker, J. (2020, November 19). Best Network Automation Tools & Software for Making your Life Easier! PC & Network Downloads - PCWDLD.Com. https://www.pcwdld.com/network-automation-tools-and-software

Samoylenko, N. (2022, January 16). Module netmiko — Python for network engineers 1.0 documentation. Pyneng.Readthedocs. Retrieved 4 April 2022, from https://pyneng.readthedocs.io/en/latest/book/18_ssh_telnet/netmiko.html

Slattery, T. (2020, January 16). Compare Paramiko, Netmiko and NAPALM network automation. SearchNetworking. https://searchnetworking.techtarget.com/tip/Network-automation-with-Python-Paramiko-Netmiko-and-NAPALM.

Zadka M. (2019) Paramiko. In: DevOps in Python. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-4433-3_9

9 Anexos

9.1 Rutinas de comandos.

- Automatización de backups
  Rutina de comando(s):
  - Copy running-config tftp
  - "IP server TFTP"
  - "File name"
  - "\n"
- Creación, configuración y cambio de VLANs en puertos de acceso y troncales.
  Rutina de comando(s):
  - Vlan "vlan ID"
  - Name "name"
  - Interface "interface type" "interface slot and number"
  - Switchport mode access / switchport mode trunk
  - Switchport access vlan "vlan ID" / switchport trunk allowed vlan add "vlan ID"
- Configuración de protocolo de árbol de expansión (STP)
  Rutina de comando(s):
  - Spanning-tree mode "mst/pvst/rapid-pvst"
  - spanning-tree vlan "vlan ID" priority "priotiry number"
- Configuración de políticas de navegación en equipo de perímetro firewall
  Rutina de comando(s):
  - config firewall policy
  - edit 0
  - set name "Policy name"
  - set srcintf "Source interface"
  - set dstintf "Destination interface"
  - set srcaddr "IP object/group"
  - set dstaddr "IP object/group"
  - set action "accept/deny"
  - set service "Service object/group"
  - set inspection-mode "Proxy/flow"
- Configuración de port-security
  Rutina de comando(s):
  - Interface "interface type" "interface slot and number"
  - Switchport port-security maximun "MAC max addresses"
  - Switchport port-security violation "protect/restrict/shutdown"
  - Switchport port-security aging time "time in sec"
  - Switchport port-security aging type inactivity
  - Switchport port-security mac-address sticky

    - Switchport port-security
    - Spanning-tree bpduguard enable
    - Spanning-tree bpduguard enable
    - Spanning-tree link-type point-to-point
    - Spanning-tree portfas enable
- Configuración de *timing* a políticas de navegación

  Rutina de comando(s):
    - config firewall policy
    - set schedule "schedule object"
- Configuración de protocolo 802.1X

  Rutina de comando(s):
    - switchport access vlan "vlan ID"
    - switchport mode access
    - authentication control-direction in
    - authentication event fail action authorize vlan "unauthorized vlan ID"
    - authentication event server dead action authorize vlan "authorized vlan ID"
    - authentication event server dead action authorize voice
    - authentication event no-response action authorize vlan "unauthorized vlan ID"
    - authentication event server alive action reinitialize
    - authentication host-mode multi-domain
    - authentication order dot1x mab
    - authentication priority dot1x mab
    - authentication port-control auto
    - authentication periodic
    - authentication timer reauthenticate server
    - mab
    - dot1x pae authenticator
    - dot1x timeout tx-period 10
    - dot1x timeout supp-timeout 5

9.2 Código de aplicativo (script).

```
from os import read
from tkinter import *
from tkinter import messagebox
import tkinter
from tkinter.ttk import Combobox, Notebook
from typing import Sized
from netmiko import *
from tkinter import filedialog

#from typing_extensions import ParamSpec

#-------------------------------------------------------------------------------------

class DEVICEIOS:

    def __init__(self, ipdir, user, pas, dtype) -> None:

        cisco_tt = {

            'device_type': dtype,
            'host':   ipdir,
            'username': user,
            'password': pas

        }

        try:
            self.net_connect = ConnectHandler(**cisco_tt)
        except NetMikoAuthenticationException:
            UI.pop_window(self, "Authentication error")
        except NetMikoTimeoutException:
            UI.pop_window(self, "Unreachable device")

    def show_x_ios(self, show_command):

        output = self.net_connect.send_command(show_command)
        return output

    def conf_x_ios(self, conf_command):

        output = self.net_connect.send_config_set(conf_command)
        return output

#-------------------------------------------------------------------------

class UI(Frame):
    """Docstring."""

    def __init__(self, parent=None):
        Frame.__init__(self, parent)
        self.parent = parent

        self.init_ui()
        self.info

    def pop_window(self, msgerror):
        messagebox.showerror("Error", msgerror)

    def button_test_show(self, comando):

        self.txt_device_ios_show.delete(1.0, END)       # Limpia cuadro de texto sección show
        test= self.DEVICE_X.show_x_ios(comando)
        self.txt_device_ios_show.insert(INSERT, test)
```

```
    def button_test_conf(self, comando):

        self.txt_device_ios_conf.delete(1.0, END)        # Limpia cuadro de texto sección show
        test= self.DEVICE_X.conf_x_ios(comando)
        self.txt_device_ios_conf.insert(INSERT, test)

#--------------------MÉTODOS SECCION SHOW-----------------------------------------------

    def lista_comandos_show_update(self,data):

        self.command_list_sh.delete(0, END)
        for item in data:
            self.command_list_sh.insert(END, item)

    def lista_comandos_show_fillout(self, e):

        # Delete whatever is in the entry box
        self.command_entry_sh.delete(0, END)

        # Add clicked list item to entry box
        self.command_entry_sh.insert(0, self.command_list_sh.get(ANCHOR))

    def lista_comandos_show_check(self, e):

        # grab what was typed
        typed = self.command_entry_sh.get()

        if typed == '':
            toppings = self.vari_a
        else:
            toppings = []
            for item in self.vari_a:
                if typed.lower() in item.lower():
                    toppings.append(item)

        # update our listbox with selected items
        self.lista_comandos_show_update(toppings)

#--------------------MÉTODOS VISTAS SECCION CONF-----------------------------------------

    def commands_conf(self,e):

        if e=="Create VLAN":
            self.btn1= Button(self.tab_3, anchor="center",text="intentando")
            self.btn1.grid(row=4, column=0)
        else:
            self.btn1.grid_remove()

#--------------------MÉTODOS SECCIÓN CONF-----------------------------------------------

    def lista_comandos_conf_update(self,data):

        self.command_list_conf.delete(0, END)
        for item in data:
            self.command_list_conf.insert(END, item)

    def lista_comandos_conf_fillout(self, e):

        # Delete whatever is in the entry box
        self.txt_device_ios_conf.delete(1.0, END)
        self.command_entry_conf.delete(0, END)

        # Add clicked list item to entry box
        self.command_entry_conf.insert(0, self.command_list_conf.get(ANCHOR))

        self.commands_conf(self.command_list_conf.get(ANCHOR))
```

```
    def lista_comandos_conf_check(self, e):

        # grab what was typed
        typed = self.command_entry_conf.get()

        if typed == '':
            toppings = self.vari_b
        else:
            toppings = []
            for item in self.vari_b:
                if typed.lower() in item.lower():
                    toppings.append(item)

        # update our listbox with selected items
        self.lista_comandos_conf_update(toppings)

    def open_your_conf_script(self):

        self.command_entry_conf.delete(0, END)

        text_file = filedialog.askopenfilename(initialdir="/home/tele", title="Open Text File", filetypes=(("Text Files", "*.txt"), ))

        text_file = open(text_file, 'r')
        stuff = text_file.read().split("\n")

        self.txt_device_ios_conf.delete(1.0,END)
        self.txt_device_ios_conf.insert(INSERT, stuff)
        text_file.close()

        self.config_commands=stuff

        self.command_entry_conf.delete(0, END)
        self.command_entry_conf.insert(0, "PERSONALIZED")

        self.bot_device_ios_conf_go["state"]=NORMAL

#-------------------------------------------------------------------------------------

    def init_ui(self):

        """Aquí colocamos los widgets."""
        self.parent.title("Automatice su Red")

        self.tab_control = Notebook(self.parent)

    #------------------PANTALLA PPAL-------------------------------------

        tab_1 = Frame(self.tab_control)

        self.tab_2 = Frame(self.tab_control)
        self.tab_3 = Frame(self.tab_control)
        self.tab_control.add(tab_1, text='Principal')

        self.tab_control.pack(expand=1, fill='both')

        lbl_1 = Label(tab_1, text="Hola \n Seleccione Sistema Operativo del Dispositivo", font=("Arial Bold", 20))
        lbl_1.pack(pady=40)

        ip_label= Label(tab_1, text="Ingrese Ip de dispositivo")
        ip_label.pack()
        self.ip_addr_txt=Entry(tab_1)
        self.ip_addr_txt.pack(pady=10)

        user_label= Label(tab_1, text="Ingrese usuario de dispositivo")
        user_label.pack()
        self.user_addr_txt=Entry(tab_1)
        self.user_addr_txt.pack(pady=10)
```

```
        pass_label= Label(tab_1, text="Ingrese password de dispositivo")
        pass_label.pack()
        self.pass_addr_txt=Entry(tab_1, show="*")
        self.pass_addr_txt.pack(pady=10)

        bot_cisco= Button(tab_1, text="Cisco IOS", bg="orange", fg="red", width=13, command= lambda:
self.device_ios("cisco_ios"))
        bot_cisco.pack(pady=10)

        bot_foti= Button(tab_1, text="Fortigate IOS", bg="orange", fg="red", width=13, command= lambda:
self.device_ios("fortinet"))
        bot_foti.pack(pady=10)

        bot_aruba= Button(tab_1, text="Aruba IOS", bg="orange", fg="red", width=13, command= lambda:
self.device_ios("aruba"))
        bot_aruba.pack(pady=10)

        bot_3com= Button(tab_1, text="3COM IOS", bg="orange", fg="red", width=13, command= lambda:
self.device_ios("3com"))
        bot_3com.pack(pady=10)

    def device_ios(self, DT):

        self.ip_addr=self.ip_addr_txt.get()
        self.user_addr=self.user_addr_txt.get()
        self.pass_addr=self.pass_addr_txt.get()

        if DT == "cisco_ios":
            self.DEVICE_X = DEVICEIOS(self.ip_addr, self.user_addr, self.pass_addr, DT)
        if DT == "fortinet":
            self.DEVICE_X = DEVICEIOS(self.ip_addr, self.user_addr, self.pass_addr, DT)

    #----------------------------PANTALLA 2 ----------------------------------------------------------------------

        self.tab_control.add(self.tab_2, text='Show Commands')
        self.tab_control.add(self.tab_3, text='Config Commands')

        self.tab_2.grid_rowconfigure(0, weight=1)
        self.tab_2.grid_columnconfigure(0, weight=1)

        show_label= Label(self.tab_2, anchor="center",text="Busque o ingrese comando show")
        show_label.grid(row=0, column=0)

        self.command_entry_sh=Entry(self.tab_2,)
        self.command_entry_sh.grid(row=1,column=0, pady=10, sticky="ew")

        bot_device_ios_show_go = Button(self.tab_2, text="Enviar", bg="orange", fg="red", width=8, command= lambda:
self.button_test_show(self.command_entry_sh.get()))
        bot_device_ios_show_go.grid(row=1, column=1, sticky="w")

        self.command_list_sh= Listbox(self.tab_2,height=5)
        self.command_list_sh.grid(row=2,column=0, sticky="new")

        scrollb_1 = Scrollbar(self.tab_2, command=self.command_list_sh.yview)
        scrollb_1.grid(row=2, column=1,sticky="wns") # Verificar funcionalidad del parámetro "sticky=nsew"
        self.command_list_sh.configure(yscrollcommand=scrollb_1.set)

        #---------------------------------------------------------------
        with open('/home/tele/Documentos/COMMANDS/Comandosshow2.txt','r') as f:
            self.vari_a = f.read().split("\n")
        #---------------------------------------------------------------

        self.lista_comandos_show_update(self.vari_a)
        self.command_list_sh.bind("<<ListboxSelect>>", self.lista_comandos_show_fillout)
        self.command_entry_sh.bind("<KeyRelease>", self.lista_comandos_show_check)
```

```
        self.txt_device_ios_show = Text(self.tab_2, wrap=WORD)
        self.txt_device_ios_show.grid(row=3, column=0, columnspan=5, sticky="nswe")

        # create a Scrollbar and associate it with txt
        scrollb = Scrollbar(self.tab_2, command=self.txt_device_ios_show.yview)
        scrollb.grid(row=3, column=1, sticky="nse")
        self.txt_device_ios_show.configure(yscrollcommand=scrollb.set)

    #-----------------PANTALLA 3-------------------------------------------------------

        self.tab_3.grid_rowconfigure(0, weight=5)
        self.tab_3.grid_rowconfigure(1, weight=1)
        self.tab_3.grid_rowconfigure(2, weight=1)
        self.tab_3.grid_rowconfigure(3, weight=1)
        self.tab_3.grid_columnconfigure(0, weight=1)

        conf_label= Label(self.tab_3, anchor="center",text="seleccione config a realizar")
        conf_label.grid(row=0, column=0,columnspan=2,sticky="nwe")

        self.command_entry_conf=Entry(self.tab_3,)
        self.command_entry_conf.grid(row=1,column=0, pady=10, sticky="ew")

        self.config_commands=[]

        self.bot_device_ios_conf_go = Button(self.tab_3, text="Enviar", bg="orange", fg="red", width=8, command=
lambda: self.button_test_conf(self.config_commands))
        self.bot_device_ios_conf_go.grid(row=1, column=3, sticky="w")
        self.bot_device_ios_conf_go["state"]=DISABLED

        self.command_list_conf= Listbox(self.tab_3,height=5)
        self.command_list_conf.grid(row=2,column=0,sticky="new")

        scrollb_2 = Scrollbar(self.tab_3, command=self.command_list_conf.yview)
        scrollb_2.grid(row=2, column=1,sticky="wns")
        self.command_list_conf.configure(yscrollcommand=scrollb_2.set)

        self.bot_device_ios_conf_select = Button(self.tab_3, text="Buscar", bg="orange", fg="red", width=8, command=
self.open_your_conf_script)
        self.bot_device_ios_conf_select.grid(row=2, column=3, sticky="we")

        #----------------------------------------------------------------
        with open('/home/tele/Documentos/COMMANDS/listaconfig.txt','r') as g:
            self.vari_b = g.read().split("\n")
        #----------------------------------------------------------------

        self.lista_comandos_conf_update(self.vari_b)
        self.command_list_conf.bind("<<ListboxSelect>>", self.lista_comandos_conf_fillout)
        self.command_entry_conf.bind("<KeyRelease>", self.lista_comandos_conf_check)

        self.txt_device_ios_conf = Text(self.tab_3, wrap=WORD)
        self.txt_device_ios_conf.grid(row=3, column=0, columnspan=2,sticky="nswe")

        # create a Scrollbar and associate it with txt
        scrollb_3 = Scrollbar(self.tab_3, command=self.txt_device_ios_conf.yview)
        scrollb_3.grid(row=3, column=2,sticky="ns") # Verificar funcionalidad del parámetro "sticky=nsew"
        self.txt_device_ios_conf.configure(yscrollcommand=scrollb_3.set)

if __name__ == "__main__":

    ROOT = Tk()
    ROOT.geometry("700x600")
    APP = UI(parent=ROOT)
    APP.mainloop()
    ROOT.destroy()
```