Python basic:

Single line comment : #
Multiline Comment:

```
'''This is a comment
written in
more than one line
'''

print("Hello World")
```

2nd ex:

```
'''
This is Network base command to get interface information
like IP address and port
connected/not connected output.
'''
Output = 'show ip int bri'
print(Output)
```

```
Basic Data type:
Python have different data type


Fist is

Number:

Integer == int()
Boolean value == bool()

Float --- float()
```

```
Sequence:

string ==str()
List == list()
Tuple -- tuple()

Mapping:

Dictionary == dict()
```

```
# Python String:

Python String

# print('Vlan_Name')
# print("Vlan_Name")


Assign string to a variable:
Vlan_ID='200'
print(Vlan_ID)

Vlan_ID= "200"
print(Vlan_ID)
```

```
Multiline String:

Command = '''
int gi0/1
desc WAN Link
no switchport
ip address 10.1.1.1 255.355.255.0
'''

print(Command)
```

```
# Variables:
```

```
Variables are container for storing data value
to declare the name of variable use equal or assignment operator = to
assign the value

no need to declare the variable type
```

Types language

1) Statically type
   Need to specify data type of variable
   Eg: c, c++,java
2) Dynamically typed
:

   No need to specify data type of variable
   Interpreter assign the data type to a variable at runtime
   Eg: python,Ruby

——----------

Create Variable: A variable is created at the moment to assign a value.

```
# Device_Name = "Router1"

# Device_port = "22"

Vlan_200,Vlan_300 = "Prod","MGMT"
print(Vlan_200)
print(Vlan_300)
```

```
# We can assign the same value to multiple variables
host = IP_address= "192.168.1.1"

print(host)
print(IP_address)
```

```
# Variables rule :

# 1) start with letter or underscore
```

```
# Vlan_ID = "100"
# _VlanID= "100"
# print(_VlanID)
# print(Vlan_ID)
# 2) Can not start with number

12Valn = "12"
print(12Vlan)

3) it contain only alphanumeric char and underscore

4) variable name are case sensitive

vlan_id = "200"
VLAN_ID= "300"

5) variable name cannot be any type of python keywords
```

Techniques to define variable name:

```
Camel case= each word except the first starts with capital
letter

vLanName= "prod"

Pascal case = Each word start with capital letter

VlanName= "prod"

3) Snake Case:  each word separated by underscore

vlan_name = "prod"
```

Python Number :

```
# 1) int
# 2) float
```

```
# 3) complex

a = 20 # inter
b= 20.4 #float
c = 2 +3j #complex

print(type(a))
print(type(b))
print(type(c))
```

```
# Boolean Values :
```

```

# print(15>10)
print(15==10)
print(15<10)
```

```
# None type:
```

```
1) None keyword is used to define a null value
2) It is not same a 0, False, or empty string
3) None is a data type of its own and Only None can be None.

x = None

print(x)
print(type(x))
```

```
# print(IP.split())


# Split Method:

# IP.split(separator, maxsplit)

# Maxsplit: Specifies how many splits to do.
```

```
# it default value is -1 that means "all occurence"

# Command = "Show ip interface brief,show run"
# x = Command.split(",")
# print(x)
# y = Command.split()
# print(y)

Device= "Cisco#Juniper#Arsita#Dell"
# Use #
# Output == ["cisco","juniper","Arista","Dell"]


x = Device.split("#")
print(x)
```

```
# IP= "192.168.10.254"



# print(IP.split())


# Split Method:

# IP.split(separator, maxsplit)

# Maxsplit: Specifies how many splits to do.
# it default value is -1 that means "all occurence"

Command = "Show ip,interface brief,show run"
x = Command.split(",")
print(x)
y = Command.split(",",1)
print(y)

# Device= "Cisco#Juniper#Arsita#Dell"
# # # # Use #
```

```
# # # # Output == ["cisco","juniper","Arista","Dell"]


# x = Device.split("#",2)
# print(x)
```

Type Casting:

It helps to change the variable value into a different data type.

```
VLAN_ID = "100"

print(type(VLAN_ID))
VLAN_ID_100= int(VLAN_ID)
print(type(VLAN_ID_100))
Vlan = float(VLAN_ID)
print(type(Vlan))
```

Note : if we have decimal point in the string then directly we can not convert in int.

Solution : First convert in float then in int.

```
VLAN_ID = "250.1"

print(type(VLAN_ID))

vlan = float(VLAN_ID)
print(int(vlan))
```

Useful Build-in function:

```
print()===> This function prints the specified msg to the screen.
```

Or it display our output

len() function==> return number of characters in that string or number of items/value in an object.

Eg:

Command = "Show bgp nei"

Output = len(Command)
print(Output)

Or
print(len(Command))

Input() function: most important
Ask to user to give input

Help to get input from user

```
Username= input("PLease Enter your Username: ")
Password= input("Please Enter your Password: ")

print(Username)
```

id() function: this returns a unique id for specific object.

This is assigned to the object when it is created.

This is the memory address of the object.

```
vlan = 100
print(id(vlan))
```

Python List: List are used to store multiple items in a single variable. List items/values are ordered, changeable and allow duplicate value. Use square brackets [ ] to create list.

Ordered means that items/value have define order and it can not change.

Ex:

```
Vendor = ["Cisco","Juniper","Arista"]
```

Access Items:

To access items we can refer to the index of that items.

```
Vendor = ["Cisco","Juniper","Arista"]     # List of devices.

print(Vendor[2])
```

Negative Index:

Use -1 for the last items.

```
Command= ["show inter dis","show ip route","show bgp sum","show run","show
ntp status"]
print(Command[4])

print(Command[-1])  #negtive index
```

```
#Insert Method: To insert a new items without replacing any of the
existing values.we can use inset() method.
Device_list = ["cisco","Juniper","Arista"]
```

```
Device_list.insert(2,"HP")

print(Device_list)
```

Append method:

```
#Append method: use to add item at the end of the list.
# Device_list = ["cisco","Juniper","Arista"]

# Device_list.append("HP")

# print(Device_list)

vlan_name = ["mgmt","prod","Tech","Agg"]
print(vlan_name)
vlan_name.append("guest")
print(vlan_name)
```

```
#Remove Method: removes the specified item from list.
# vlan_name = ["mgmt","prod","Tech","Agg"]

# vlan_name.remove("Tech")

# print(vlan_name)

#POP Method: it removes the items with specified index.


vlan_name = ["mgmt","prod","Tech","Agg"]

vlan_name.pop(2)
print(vlan_name)

Note: if you do not specify the index then pop() method remove last item.
```

```
#del keyword. this keyword also remove the specified index.

vlan_name = ["mgmt","prod","Tech","Agg"]

del vlan_name[0]
print(vlan_name)
```

List Slicing:

Help to access a range of items/element/value in a list.

We can specify the start index and end index, separated by colon(:), to return a part of list.

The format of list slicing is

[start: end:step]

start : it is index of the list where slicing start
end : stop is the index where slicing ends
Step: it allows us to select nth item within the range start to and default step is 1
stop(end)

Ex:

```
vlan_name = ["mgmt","prod","guest","Tech","Agg"]



print(vlan_name[:4])
```

```
vlan =["1","30","11","3","10","20","30","100"]

print(vlan[1:6:1])
print(vlan[1:6])
```

Python Tuples:
Use to store multiple value in single variable.

Tuple is ordered, unchangeable and allow duplicate value.
Tuple items are indexed.

Tuple is immutable because we can not change its value when we create a tuple.

To create Tuple use ()

To create tuple with single items then we have to use comm(,) after item.
Ex:

```
Netmask = ("255.255.255.255",)
```

Ex: `device_type= ("Cisco","Juniper","Arista")`

```


print(type(device_type))
```

Access Items: We can use index number

```
device_type= ("Cisco","Juniper","Arista")

#print(device_type[1])

device_type[1]="HP"
print(device_type)
```

Count Method:

```
# Count Method: Use to find how many elements match that value that you
pass in.

VLAN_ID= ("2",3,4,"2","6","2")

print(VLAN_ID.count("6"))
# Count Method: Use to find how many elements match that value that you
pass in.

VLAN_ID= ("2",3,4,"2","6","2")

# print(VLAN_ID.count("6"))

#Index method: It help to find the index number of any items.

print(VLAN_ID.index(4))
```

Python Set:

Store multiple items in a single variable.

Set items are unordered, unchangeable and do not allow duplicate value.

Note: Set items are unchangeable but we can add items and we can remove them.

To create set we can use {}

With the help of indexes we can not access items.

```

```

```
VLAN_ID= {"2",3,4,"2","6","2"}


print(VLAN_ID)
```

——————————————————————————————

Python Dictionaries:

Dict are used to store values in key:value pairs.

Dict is ordered, changeable, and does not allow duplicates.

To create set we can use {}

```
Device_type ={
   "Vendor":"Cisco",
   "Model":"C9500",
   "IOS":12.5
}

print(Device_type)
```

To access value we can use key

```
Device_type ={
   "Vendor":"Cisco",
   "Model":"C9500",
   "IOS":12.5
}

print(Device_type['Vendor'])
print(Device_type['Model'])
```

Note: Dic cannot have two items with same key

```
Device_type ={
   "Vendor":"Juniper",
   "Vendor2":"Cisco",
   "Model":"C9500",
   "IOS":12.5
}

print(Device_type)
```

Nested Dic: Dict inside Dict

A dict within a dict is called a nested dict.

```
Device_type ={
   "cisco":{"Router1":"South","Switch1":"West"},
   "Arista":{"Router":"East","Switch2":"North"}
}

# Output = South
# # print(Device_type)
# # print(Device_type["cisco"])
# print(Device_type["Arista"])

print(Device_type["cisco"]["Router1"])
print(Device_type["Arista"]["Switch2"])
```

——————————————————————————————

Python If and else :

If and else statement is a conditional statement in programming.

Equals: a==b

Not Equals: a!=b

Less than: a<b
Less than and equal to: a<=b
Greater Than: a>b
Greater than and Equal to: a>=b

If statement example:

```
is_up = "up"

if is_up=="up":
   print("Interface is up")
```

Elif: this is keyword if the previous statement were not true then it will try this conditions.

```
a = 40
b = 40

if b>a:
   print("b is greater than a")
elif a==b:
   print("a and b both are equal")
```

Example:

```
User = input("Enter Your Username: ")
```

```
if User == "Amitesh":
   print("You are authorized to Login:")
elif User== "Ajay":
   print("You are authorized to Login:")
else:
   print("You are not authorized to Login:")
```

```
# Else: this is keyword
# it catches anything which is not caught by if statement.
```

```
is_up ="Down"

if is_up =="up":
   print("Interface is up")
else:
   print("Interface is down")
```

```
a = 300
b = 40

if b>a:
   print("b is greater than a")
elif a ==b:
   print("b is equal to a")
else:
   print("a is grether than b")
```

Nested if: you can have if statement inside if statement. That is called nested if statement.

```
x = 50

if x>20:
   print("x is above 20")
   if x>30:
       print("X is also above 30")
```

```
    else:
        print("but not above 30")
else:
    print("not above 20")
```

Python Loops: A loop is a set of instructions or statements that are executed and repeated until a certain condition is meet.

Python have two loop commands.

1) For loop
2) While

While loop: with the while loop we can execute a set of statements as long as a condition is true.

```
i = 1

while i<5:
    print(i)
    i= i+1
```

For Loop:

Loop is used for iterating over a sequence that is either a list, a tuple, dict, a set or string).

```
Ajay = ["Cisco","Juniper","Arista","HP"]

for i in Ajay:
```

```
    print(i)
```

```
ip_list = ["192.168.10.10","192.168.10.11","192.168.10.12"]

for ip in ip_list:
    print(ip)
```

Range() Function: it is return a sequences of numbers, starting from 0(default, increment by 1 and stops before a specified number.

Syntax:
range(start,stop,step)

Ex: range(3,6) that means it will start from 3 and end before 6 it means it will end on 5 so we can say that in range stop is excluded.
Default step is 1.

```
for i in range(0,10,2):
    print(i)
```

```
# vlan (2,10)

for i in range(2,10):
    print("Vlan "+str(i))
```

```
IP = 192
mask = 255
print(f"ip address {IP} {mask}")    # Best Method  this is f string
print("IP address " + str(IP)+" "+ str(mask))
```

```
print("IP address {1} {0}".format(IP,mask))    # Second Best Method
```

Nested Loops:

A nested loop is a loop inside a loop.

It have two loop
   1) Inner loop
   2) Outer loop

```
IP_List = ["192.168.1.1","192.168.1.2","192.168.1.3"]

# ip address 192.168.1.1 255.255.255.255
# ip address 192.168.1.2 255.255.255.255
# ip address 192.168.1.3 255.255.255.255

Netmask = ("255.255.255.255","255.255.255.0")  #Tuple

for ip in IP_List:
   for mask in Netmask:
      print(f"ip address {ip} {mask}")
```

Inner loop will be executed one time for each iteration of the outer loop.

```
IP_List = ["192.168.1.1","192.168.1.2","192.168.1.3"]

# ip address 192.168.1.1 255.255.255.255
# ip address 192.168.1.2 255.255.255.255
# ip address 192.168.1.3 255.255.255.255

Netmask = ("255.255.255.255","255.255.255.0","255.255.255.252")
#Tuple

for ip in IP_List:
   for mask in Netmask:
      print(f"ip address {ip} {mask}")
```

Function:

A function is a block of code which is only run when it is called.

Def keyword use to create function.

```
def Ajay():
   print("Hello I am Ajay")

Ajay()  # Calling a Function
```

With the help of a for loop.

```
def Ajay(device):
   print(device+" Vendor")


Vendor_List = ["Cisco","Juniper","Arista"]
for device in Vendor_List:
   Ajay(device)
```

3 Aug-2024 Session

This Week Task:

Python Program to Remove First Occurrence of a Character in a String:

given_str = "Hello World"

Char = "l"

Output : Helo World

Python Program to find First Occurrence of a Character in a String
Note: Use input function to ask user to give String and Characcter.

Python program to Remove Odd Characters in a String:

given_str = "Hello World"

—---

Solution:

```
given_str = "Hello World"
```

```
Char = "l"

NewStr = ''

# 1) Slice
# 2) if or else statement
# 3) for loop

Len = len(given_str)

# print(Len)

for i in range(Len):
   if given_str[i]==Char:
       NewStr= given_str[0:i]+given_str[i+1:Len]
       break
print(NewStr)
```

——————————————————————

```
given_str = input("Please Enter the string:")

Char = input("PLeae Enter the Cha: ")

NewStr = ''

Flag = 0

# 1) if or else statement
# 2) for loop

Len = len(given_str)

# print(Len)

for i in range(Len):
   if given_str[i]==Char:
```

```
        Flag = 1
       # print("Yes, First Char found")
        break

if Flag == 1:
   print("Yes, First Char found")
else:
   print("No, First Char not found")
```

By default a function must be called with the correct number of arguments. Meaning that your function expects 2 arguments and you have to call the function with 2 arg.

If you try to call the func with less or more then it will give an error.

```
def Person(Name,age):
   print("My name is "+Name)
   print("My age is "+ str(age))

Person("Mohan",27)
```

If you do not know how many arguments that will be passed into your function. Then add * before the parameter name in the function definition.

```
def Device(*Name):
   print("The device is "+ Name[0])
   # print("My age is "+ str(age))

Device("Cisco","JUniper","Arista")
```

Python Network Modules:

Module Netmiko:

1. Netmiko is a popular python library which is designed to simplify SSH Connections to Network devices.

2. This is based on paramiko.paramiko is a python implemented ssh protocol.

3. This is Multi vendor library

4. This will provide a simple and easy to use interface for interacting with network devices.

5. Netmiko is the alternative that does not support APIs.

6. Netmiko library must be installed.this is not standard python library so we need to install it.

Use this command to install netmiko
Pip install netmiko

Supported Devices:

1) Arista vEOS
2) Cisco ASA
3) Cisco IOS XR
4) Cisco SG300
5) HP Procurve

6) Juniper junos
7) Linux

https://github.com/ktbyers/netmiko/tree/develop/netmiko

How to install Netmiko:

1) Pip install netmiko

To check use ⇒pip list command from CMD

4 Aug 2024 Session:

First Python netmiko script:

Example-1

```
from netmiko import ConnectHandler

# From netmiko import: This  statement tells python to import specific
functionality from the netmiko library

# Connectionhandler: This is the class within the netmiko which is used to
estabilsh connecion to network devices and intract with them

# Once we imported then we can use ConnectHandler to create connection
object for network devices.

#Create a dict which represents the devices.
Cisco_Devies= {
   "device_type":"cisco_ios",
   "host":"10.1.1.1", # Give your host IP address
   "username":"cisco",
```

```
   "password":"123456789",
   #"port":8021  default port is 22 we are using.
   "secret":"123456789"
}
# #Create an object to connect with devices.
Connection = ConnectHandler(**Cisco_Devies)  # **(Double Asterisk) This is
operator is used for unpacking a dict.



# use_keys(optional)==A boolean flag which indicate whether to use public
key auth(default is False)
# auth_timeout=Timeout value in sec(default is 10 Sec)

Output = Connection.send_command("show ip int bri")

print("Closing the connection")
Connection.disconnect()
```

——----------

Example-2

```
from netmiko import ConnectHandler

#Create a dict which represent the divices.
Cisco_Devies= {
   "device_type":"cisco_ios",
   "host":"10.1.1.1", # Give your host IP address
   "username":"cisco",
   "password":"123456789",
   #"port":8021  default port is 22 we are using.
   "secret":"123456789"
}
#Create a object to connect with devices.
Connection = ConnectHandler(**Cisco_Devies)  # **(Double Asterisk) This is
operator is used for unpacking a dict.

Output = Connection.send_command("show ip int bri")
print(Output)
```

```
print("Closing the connection")
Connection.disconnect()
```

Aug 11 2024

Example-3

```
#show run

from netmiko import ConnectHandler

Cisco_device = {
   "device_type":"cisco_ios",
   "host":"10.1.1.1",
   "username":"cisco",
   "Password":"cisco"
}

# This will help to make connection with Network Devices.
Device_Connect =ConnectHandler(**Cisco_device)


# This will fetch running config and save it in Running_Config_Output
Variable
Running_Config_Output = Device_Connect.send_command("show running")
#Print Will help to diaplay the Output
print(Running_Config_Output)

print("Closing Connection ")
Device_Connect.disconnect()
```

Example-3

To know the prompt of the device we can use find_prompt() Method.

```
from netmiko import ConnectHandler

cisco_device = {
```

```
    "device_type":"cisco_ios",
    "host":"10.1.1.1",
    "username":"cisco",
    "password":"123456789",
}

Device_Connect = ConnectHandler(**cisco_device)


Prompt1 = Device_Connect.find_prompt()
print(Prompt1)
Device_Connect.config_mode()
Prompt2 = Device_Connect.find_prompt()
print(Prompt2)
Config_Output= Device_Connect.send_command("hostname My_Router2")
Device_Connect.exit_config_mode()

Prompt3 = Device_Connect.find_prompt()
print(Prompt3)

# print(Running_Config_Output)

print("Closing The Connection")
Device_Connect.disconnect()
```

Example-3:

To Send NTP Command from Netmiko script

```
from netmiko import ConnectHandler

device_1= {
    "device_type":"cisco_ios",
    "host":"10.1.1.1",
    "username":"cisco",
    "password":"123456789"
}

Connection = ConnectHandler(**device_1)
```

```
Connection.config_mode()
Connection.send_command("ntp server 192.168.10.1")
Connection.exit_config_mode()
Output = Connection.send_command("show run | in ntp")
print(Output)
```

```
#Sent Multiple Show Command from Netmiko Script.then Use can achieve this with the help of for Loop
```

Example-5

```
#Sent Multiple Show Command from Netmiko Script.then Use can achieve this with the help of for Loop
```

```
from netmiko import ConnectHandler


device_1= {
   "device_type":"cisco_ios",
   "host":"10.1.1.1",
   "username":"cisco",
   "password":"123456789"
}


Connection = ConnectHandler(**device_1)


Show_Commad = ["show ip route",'Show ip int bri',"show clock","show ntp status"]


for Command in Show_Commad:
   print(f"\n **Sending Command {Command}***")
   output = Connection.send_command(Command)
   print(output)
```

```
Connection.disconnect()
```

Aug 17 2024 Session 10:

Python program to find Smallest number in a list using for Loop.

Given_list= [223,43,22,67,54]

Given_list= [223,43,22,67,54]

#Smallest Number is 22

Smallest_Number = Given_list[0]

```
for num in Given_list:
    if Smallest_Number>num:
        Smallest_Number=num
print("The Smallest number in given list is:",
Smallest_Number)
```

Anagram string: if one string forms by rearranging the other string characters, it is an anagram string.

Example : triangle and integral will form by rearranging the characters.

Python Program to Check If Two Strings are Anagram:

Note: Use input function to ask user to give String and Characcter.

```
Str1 = input("Enter the first String:")
Str2 = input("Enter the Second String: ")

Sorted_One = sorted(Str1)
print(Sorted_One)

Sorted_Second = sorted(Str2)
print(Sorted_Second)

if Sorted_One==Sorted_Second:
    print("Yes")
else:
    print("No")
```

Or

```
# if sorted(Str1)==sorted(Str2):
#     print("Two String are Anagrams")
# else:
#     print("Two String are not Anagrams")
```

Palindrome: A string could be a Palindrome string in Python if it remained the same after reversing it.

Python Program to check string is palindrome or not

```
Str = input("Please Enter your Own String:")

Output_Str = ""  #This is Empty String

for i in Str:
    Output_Str= i +Output_Str
#print(Output_Str)

if Str ==Output_Str:
    print("Yes, This is Palindrome")
```

```
else:
    print("This is not")
```

2 Method:

```
Str = "aba"


Output_Str= Str[::-1]

if Str ==Output_Str:
    print("Yes")
else:
    print("No")
```

—----------

Getpass library:

1. It hide the password
2. Getpass prompts the user for a value, password in our case without echoing what the user types to the terminal.

3. Getpass read the user input and save it as string.

Example-3
Example-6

```
from netmiko import ConnectHandler
import getpass

Passwd = getpass.getpass("Please Enter The Password: ")

device_1= {
   "device_type":"cisco_ios",
   "host":"10.1.1.4",
   "username":"cisco",
   "password":Passwd
}

Connection = ConnectHandler(**device_1)
# Connection.enable()
# Connection.config_mode()

Oupy= Connection.send_command("Sh ip route")

print(Oupy)

Connection.disconnect()
```

Send Multiple Command to a single device.

If we want to send multiple commands (a mixture of show and config commands using a single python

script. Netmiko provides a way to achieve this via the **send_config_set** method:

Send_config_set method:

1) Automatically enters to global config mode
2) After running the config, it automatically exits the global config mode.

Example-7:

```
from netmiko import ConnectHandler
import getpass


Passwd = getpass.getpass("Please Enter The Password: ")


device_1= {
  "device_type":"cisco_ios",
  "host":"10.1.1.4",
  "username":"cisco",
  "password":Passwd
}


Connection = ConnectHandler(**device_1)
# Connection.enable()
# Connection.config_mode()

Config_commads= ["interface gi0/0","desc AJAY_WAN","exit","access-list 10
permit any"]

Output = Connection.send_config_set(Config_commads)
print(Output)
```

```
Connection.disconnect()
```

18 Aug 2024 Session 11:

Python Example:

```
#Python program to remove duplicate value from list
List = [1,2,3,2,4,8,1,7,6,4,5]

First  method:
#Output = [1,2,3,4,8,7,6,5]

Empty_list = []

for i in List:
   if i not in Empty_list:
       Empty_list.append(i)
print(Empty_list)


Second method
#set ==> This does not allow dublicate vaue

# New_Set = set(List)
# print(New_Set)
# New_List = list(New_Set)
# print(New_List)
```

Example-8:

```
#send_config_from_file() ==> this help to send config from file (txt)


from netmiko import ConnectHandler
```

```
Cisco_divice= {
   "device_type":"cisco_ios",
   "host":"10.1.1.1",
   "username":"cisco",
   "password":"123456789"
}

Connection = ConnectHandler(**Cisco_divice)

print("Connection Successful ")

Config = Connection.send_config_from_file(config_file ="config.txt")

print(Config)

print(Connection.send_commad("sho up int bri"))

Connection.disconnect()
```

Exception Handling:

1. When an error occurs or exception as we call it python will stop code and it generates an error msg.
2. These exceptions we can handle using the try, except, else and finally statement.

Try : this block lets you test a block of code for error.

Except: this block lets you handle that error.

Else : this block lets you execute code when there is no error .

Finally:this block lets you execute a block of code regardless of the result of try and except block.

```
try:
print(2/1)
except:
print("We can not divide by zero")

else:
 print("Nothing went wrong ")
finally:
 print("Try and except is finished ")
```

Example-9

This is netmiko code with try and exception

```
#Using try and except to handle netmiko error like timeout, uthentication

from netmiko import ConnectHandler
from netmiko import NetMikoTimeoutException


Device_list = ["10.1.1.2","10.1.1.4","10.1.1.3"]

for device in Device_list:
   Cisco_divice= {
   "device_type":"cisco_ios",
   "host":device,
   "username":"cisco",
```

```
   "password":"123456789"
   }
   try:
            Command_list= ["show ip int bri","show ip route","sho
clock","show ntp status"] # List of Command
            print(f"\n{"#"*35}\n Connecting to device
{Cisco_divice['host']}")
            Connection = ConnectHandler(**Cisco_divice)
            print(f"\n{"#"*35}\n Connected Successfully to device
{Cisco_divice['host']}")

            for Command in Command_list:
                Output = Connection.send_command(Command)
                print(Output)


            Connection.disconnect()
   except NetMikoTimeoutException:
       print(f"Session Timeout on device {Cisco_divice["host"]}")
```

Example-10

This is Other code for Authentication Error.And we can handle this way.

```
from netmiko import ConnectHandler
from netmiko import NetMikoTimeoutException,NetMikoAuthenticationException

Cisco_divice1= {
"device_type":"cisco_ios",
"host":"10.1.1.2",
"username":"cisco",
"password":"123456789"
}
Cisco_divice2= {
"device_type":"cisco_ios",
"host":"10.1.1.3",
```

```
"username":"cisco",
"password":"12345678"
}
Cisco_divice3= {
"device_type":"cisco_ios",
"host":"10.1.1.4",
"username":"cisco",
"password":"123456789"
}
Device_list = [Cisco_divice1,Cisco_divice2,Cisco_divice3]  # This is list
of Dict


for device in Device_list:
   try:
           Command_list= ["show ip int bri","show ip route","sho
clock","show ntp status"] # List of Command
           print(f"\n{"#"*35}\n Connecting to device {device['host']}")
           Connection = ConnectHandler(**device)
           print(f"\n{"#"*35}\n Connected Successfully to device
{device['host']}")


           for Command in Command_list:
               Output = Connection.send_command(Command)
               print(Output)


           Connection.disconnect()
   except NetMikoTimeoutException:
       print(f"Session Timeout on device {device["host"]}")
   except NetMikoAuthenticationException:
       print(f"Authentication Failed on Device {device["host"]}")
```

Example-11

This is 3rd Script without importing NetMikoTimeoutException,NetMikoAuthenticationException. We can import exceptions only and we can handle all errors.

```
from netmiko import ConnectHandler
from netmiko import exceptions

Cisco_divice1= {
"device_type":"cisco_ios",
"host":"10.1.1.2",
"username":"cisco",
"password":"123456789"
}


Cisco_divice2= {
"device_type":"cisco_ios",
"host":"10.1.1.3",
"username":"cisco",
"password":"12345678"
}
Cisco_divice3= {
"device_type":"cisco_ios",
"host":"10.1.1.4",
"username":"cisco",
"password":"123456789"
}
Device_list = [Cisco_divice1,Cisco_divice2,Cisco_divice3]  # This is list
of Dict


for device in Device_list:
   try:
            Command_list= ["show ip int bri","show ip route","sho
clock","show ntp status"] # List of Command
            print(f"\n{"#"*35}\n Connecting to device {device['host']}")
            Connection = ConnectHandler(**device)
            print(f"\n{"#"*35}\n Connected Successfully to device
{device['host']}")
```

```
            for Command in Command_list:
                Output = Connection.send_command(Command)
                print(Output)


            Connection.disconnect()
    except exceptions.NetMikoTimeoutException:
        print(f"Session Timeout on device {device["host"]}")
    except exceptions.NetMikoAuthenticationException:
        print(f"Authentication Failed on Device {device["host"]}")
```

—----------

24 Aug 2024 Session 12

Python Program to find the Largest in a List

```
#Python Program to find Largest Number in a List using Loop
Number = [10, 50, 60, 120, 20, 15]

# Method
# 1) Use sort ===>to convert in assending order

# 2) max() function


# #Method-1

# print(max(Number))


# Number = [10, 50, 60, 120, 20, 15]
# largest = Number[0]
# for i in Number:
#     if i>largest:
```

```
#         largest=i
# print(f"the largest number is {largest}")

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

Number = [10, 50, 60, 120, 20, 15]
largest = Number[0]
for i in range(len(Number)):
   if Number[i]>largest:
       largest=Number[i]
print(f"the largest number is {largest}")
```

Python Program to find the Largest and Smallest Number in a List

```
Number = [10, 50, 60, 120, 20, 15]

# print(min(Number))
# print(max(Number))

largest = Smallest = Number[0]
for i in range(len(Number)):
  if Number[i]>largest:
      largest=Number[i]
  if Smallest>Number[i]:
      Smallest=Number[i]
print(f"the largest number is {largest} and Smallest Number is
{Smallest}")
```

Python File open

File handing is an important part of web application

Python has several functions for creating, deleting, reading and updating

open () function:

It takes two parameter 1) filename

2) Mode

Mode: exist 4 mode

“r”:-> read -> Default value and it means it opens a file for reading and if the file does not exist then it will give an error.

“a”:--> append → OPen a file for appending and if file does not exist then it will create a file.

“w”---> Write→ It opens a file for writing and if file does not exist then it will create a file.

“x”--> Create → Create a specific file and return an error if the file already exists.

“t”--> text –Default value → text Mode
“b”--> Binary → Binary Mode(example : image)

Syntax:

Backup = open(“Device_Confi.txt”)

Or

Backup = open(“Device_Config.txt”,”rt”)

Note: To open a file for reading it is enough to specify the name of the file

```
#OPen a file
Backup = open("Device_Config.txt","r")
print(Backup.read())
```

```
Note: If file is located in a diffrent location then we need to specify
the full path

#OPen a file
Backup = open("/home/netautoclass/Config/Device_Config.txt","r")
print(Backup.read())
```

```
from netmiko import ConnectHandler
import getpass

passwd = getpass.getpass('Please enter the password: ')

cisco_01 = {
    "device_type": "cisco_ios",
    "host": "10.1.1.4",
    "username": "cisco",
    "password": passwd, # Log in password from getpass
    "secret": passwd # Enable password from getpass
}

connection = ConnectHandler(**cisco_01)
connection.enable()  #Moving in Enable Mode

output = connection.send_command('show run')
print(output)
prompt = connection.find_prompt()
print(prompt)
hostname = prompt[0:-1]
print(hostname)

DeviceBackup = f'{hostname}-backup.txt'

with open(DeviceBackup,'w') as backup:
     backup.write(output)
     print(f'Backup of {hostname} done')
     print('#'*30)
```

```
print('Closing Connection')
connection.disconnect()
```

Script to Save file in different locations if we are working with windows system.

```
from netmiko import ConnectHandler
import getpass

passwd = getpass.getpass('Please enter the password: ')

cisco_01 = {
    "device_type": "cisco_ios",
    "host": "10.1.1.4",
    "username": "cisco",
    "password": passwd, # Log in password from getpass
    "secret": passwd # Enable password from getpass
}

connection = ConnectHandler(**cisco_01)
connection.enable()  #Moving in Enable Mode

output = connection.send_command('show run')
print(output)
prompt = connection.find_prompt()
print(prompt)
hostname = prompt[0:-1]
print(hostname)

DeviceBackup = f'C:\\Users\\Ruby\\Documents\\{hostname}-backup.txt'

with open(DeviceBackup,'w') as backup:
     backup.write(output)
     print(f'Backup of {hostname} done')
     print('#'*30)
print('Closing Connection')
connection.disconnect()
```

25 Aug 2024 Session 13

```
#List items at even number

evList = [3,6,9,11,13,15,17,19]

# Len = len(evList)
# print(Len)

# print(evList[2::2])

# for i in range(1,len(evList)):
#     print(i)
#     if (i)%2==0:
#       print(evList[i])
```

```
#Python Program to Print List Items in Reverse Order
Number = [10, 50, 60, 120, 20, 15]

#Output = [15,20,120,60,50,10]

New_List = [] #Empty List

for i in range(len(Number)-1,-1,-1):
   print(i)
   New_List.append(Number[i])
   print(New_List)
```

Take backup of multiple devices:

```
from netmiko import ConnectHandler

with open('device.txt','r') as hostname:
    devcies = hostname.read().splitlines()
```

```
    #  print(devcies)

for IP in devcies:

    cisco_01 = {
        "device_type": "cisco_ios",
        "host": IP,
        "username": "cisco",
        "password": 123456789, # Log in password from getpass
        "secret": 123456789 # Enable password from getpass
    }

    connection = ConnectHandler(**cisco_01)
    connection.enable()  #Moving in Enable Mode

    output = connection.send_command('show run')
    print(output)
    prompt = connection.find_prompt()
    print(prompt)
    hostname = prompt[0:-1]
    print(hostname)


    DeviceBackup = f'C:\\Users\\Ruby\\Documents\\{hostname}-backup.txt'


    with open(DeviceBackup,'w') as backup:
        backup.write(output)
        print(f'Backup of {hostname} done')
        print('#'*30)
    print('Closing Connection')
    connection.disconnect()
```

This script is good but not perfect because lets suppose you will run again tomorrow then it will be overwritten but we need to maintain a version. So we need date and time to maintain daily backup.

```
from netmiko import ConnectHandler
from datetime import datetime

# # getting the current time as a timestamp
# start = time.time()
with open('devices.txt') as hostname:
    devices = hostname.read().splitlines()

for ip in devices:

    cisco_01 = {
           'device_type': 'cisco_ios',
           'host': ip,
           'username': 'cisco',
           'password': '123456789',
           'port': 22,             # optional, default 22
           'secret': '123456789',      # this is the enable password
           }
    connection = ConnectHandler(**cisco_01)
    print('Entering the enable mode...')
    connection.enable()

    output = connection.send_command('show run')
    # print(output)

    # creating the backup filename (hostname_date_backup.txt)
    prompt = connection.find_prompt()
    hostname = prompt[0:-1]
    # print(hostname)

    # getting the current date (year-month-day)
    now = datetime.now()
    year = now.year
    month = now.month
    day = now.day
```

```
    hours= now.hour
    minute= now.minute

    # creating the backup filename (hostname_date_backup.txt)
    Backupfile =
f'C:\\Users\\Ruby\\Documents\\{hostname}_{year}-{month}-{day}-{hours}-{min
ute}_backup.txt'

    # writing the backup to the file
    with open(Backupfile, 'w') as backup:
        backup.write(output)
        print(f'Backup of {hostname} completed successfully')
        print('#' * 30)


    print('Closing connection')
    connection.disconnect()
```

Multithreading: Multithreading in netmiko can enhance performance of our Network script allowing you to execute tasks concurrently and improve efficiency.

```
from netmiko import ConnectHandler
from datetime import datetime
import time
import threading  #implements threading in Python

# getting the current time as a timestamp
start = time.time()

# this function backups the config of a router
# this is the target function which gets executed by each thread
def backup(device):
    connection = ConnectHandler(**device)
    print('Entering the enable mode...')
    connection.enable()
```

```
    output = connection.send_command('show run')
    # print(output)
    prompt = connection.find_prompt()
    hostname = prompt[0:-1]
    # print(hostname)

    now = datetime.now()
    year = now.year
    month = now.month
    day = now.day
    hours= now.hour
    minute= now.minute

    Backupfile =
f'C:\\Users\\Ruby\\Documents\\{hostname}_{year}-{month}-{day}-{hours}-{min
ute}_backup.txt'
    with open(Backupfile, 'w') as backup:
        backup.write(output)
        print(f'Backup of {hostname} completed successfully')
        print('#' * 30)

    print('Closing connection')
    connection.disconnect()

with open('devices.txt') as f:
    devices = f.read().splitlines()

# creating an empty list (it will store the threads)
threads = list()
for ip in devices:
    device = {
           'device_type': 'cisco_ios',
           'host': ip,
           'username': 'cisco',
           'password': '123456789',
           }
    # creating a thread for each router that executes the backup function
    th = threading.Thread(target=backup, args=(device,))
    threads.append(th) # appending the thread to the list
```

```
# starting the threads
for th in threads:
    th.start()

# waiting for the threads to finish
for th in threads:
    th.join()

end = time.time()
print(f'Total execution time:{end-start}')
```

31 Aug 2024 Session 14:

```
#Take Backup of devices with CSV:

from netmiko import ConnectHandler
import csv
import time

#Created a dic representing the devices details.

Cisco_device = {
   "device_type":"cisco_ios",
   "host":"10.1.1.1",
   "username":"cisco",
   "password":"123456789",
}



# **(Double Asterisk)  meaning --> This operator is used to unpacking a
dictionary
#This creates a connection object that establishes connection to the
network device.
Connection = ConnectHandler(**Cisco_device)
Connection.enable()  #Moving the enable

#-
```

```
output = Connection.send_command("show run")
print(output)

prompt =Connection.find_prompt()
hostname = prompt[0:-1]

DeviceBackup = f"{hostname}-backup.csv"

with open (DeviceBackup,'w') as backup:
   write= csv.writer(backup)
   write.writerow(["Command"])
   for line in output.splitlines():
       write.writerow([line])

print("Closing the connection")

Connection.disconnect()  # This helps to exit from the device.
```

Question: We have write code which take config from csv and push to device.

Hint : Use open func with read mode.

——----------

OSPF and interface but base on user choice if user will choice 1 then OSPF will config and if user will use number 2 then interface will config and they will put any other number script will notify to user.

```
from netmiko import ConnectHandler

R1 = '10.1.1.1'
R2 = '10.1.1.2'

Router_List = [R1,R2]

Username = input('Please Enter Your Username:')
```

```
Password = input('Please Enter Your Password:')

#This is function to config interface
def interface_config():
    int_name = input('Please Enter The Interface Name:')
    ip_address = input('Please Enter the IP address:')
    sub_mask = input('Please Enter The Subnet Mask:')


    commands = ['interface ' + int_name, 'ip address ' + ip_address + ' '
+ sub_mask]
    int_loopback = SSH.send_config_set(commands)
    print(int_loopback)


    int_brief = SSH.send_command('show ip inter bri')
    print(int_brief)


# This is function to config OSPF
def ospf():
    ospf_proc_id = input('Please Enter Your OSPF Process ID:')
    ospf_router_id = input('Enter Your Router ID:')
    ospf_network_count = int(input('How Many Network To Advertise:'))


    for router in range(0, ospf_network_count):
        ospf_network_id = input('Enter Your Network ID:')
        ospf_wildcard_mask = input('Enter Your wildcard mask:')
        ospd_area_id = input('Enter Your Area ID:')
        command = ['router ospf ' + ospf_proc_id,
                   'router-id ' + ospf_router_id,
                   'network ' + ospf_network_id + ' ' + ospf_wildcard_mask
+ ' ' + 'area ' + ospd_area_id]
        ospf_confi = SSH.send_config_set(command)
        print(ospf_confi)


User_Choice1 = input('Welcome to Config Utility\n1. Interface Config\n2.
OSPF Config\nPlease Make a Choice(1/2): ')
```

```
if User_Choice1 == '1':
    for router in Router_List:
        device_temp = {
            'device_type': 'cisco_ios',
            'host': router,
            'username': Username,
            'password': Password,
            'secret':Password
        }

        SSH = ConnectHandler(**device_temp)
        SSH.enable()
        print('#' * 75 + '\n' + 'Connecting To Router ' + router + '\n' +
'#' * 75 + '\n')
        interface_config()  #To CALL function  Interface Config
elif User_Choice1 == '2':
    for router in Router_List:
        device_temp = {
            'device_type': 'cisco_ios',
            'host': router,
            'username': Username,
            'password': Password,
            'secret':Password
        }

        SSH = ConnectHandler(**device_temp)
        SSH.enable()
        print('#' * 75 + '\n' + 'Connecting to Router ' + router + '\n' +
'#' * 75 + '\n')
        ospf() #To CALL function  OSPF Config
else:
    print('Invalid Input Detected\nPlease Try Again\nThank You...!!!')
```

7 Sep 2024 Session 15:

NAPALM: (Network automation and programmability abstraction layer with Multivendor support):

It is a python library that implements a set of functions to interact with different network devices using a unified API.

It supports several methods to connect to the devices, manipulate configuration and retrieve data.

—----------

Supported Network Devices.

Cisco IOS

Cisco IOS-XR

CISCO NX-OS

Arista EOS

JUniper JunOS

This is useful in environments where multiple vendors are used because it removes the need to have separate scripts for each device type.

Common interface and Mechanisms:

It provides a common interface and mechanisms to push configuration and retrieve state data from network devices.

Benefit: it allows us to manage a set of devices independently of their network OS.

For Example: We can use the function to retrieve the ARP table for Cisco IOS and Juniper Junos devices .This simplifies the automation process because we do not need to know the specific commands to retrieve data or config from devices for each OS.

Available functions/Method: we will get in this URL

https://napalm.readthedocs.io/en/latest/support/index.html

Installation: Pip install napalm

Example 1:

Obtain ARP table for single device.

```
from napalm import get_network_driver
import getpass

passwd = getpass.getpass("Please Enter the Password: ")


Driver = get_network_driver("ios")

Switch_01= {
    "hostname":"10.1.1.1",
    "username":"cisco",
    "password":passwd,
    "optional_args":{"secret":passwd}
}

Connection= Driver(**Switch_01)
Connection.open()    #Opens the connection to the network devices
print(f"Connecting to {Switch_01["hostname"]}")

output = Connection.get_arp_table() # To fetch ARP table
print(output)
Connection.close()
```

Example-2:

Convert output in Json format:

```
from napalm import get_network_driver
import getpass
import json
passwd = getpass.getpass("Please Enter the Password: ")

Driver = get_network_driver("ios")

Switch_01= {
    "hostname":"192.168.1.9",
    "username":"cisco",
    "password":passwd,
    "optional_args":{"secret":passwd}
}

Connection= Driver(**Switch_01)
Connection.open()    #Opens the connection to the network devices
print(f"Connecting to {Switch_01["hostname"]}")

output = Connection.get_arp_table() # To fetch ARP table
#print(output)
#print(output[0])    #To Access the index 0 items
#print(output[0]["mac"]) # To Access specific mac address
Connection.close()

ouput_json = json.dumps(output,indent=4) # dumps is use to convert python
output to jSON format
print(ouput_json)
```

Let's break code line by line:

From napalm import get_network_driver: this line import get_network_drive from napalm library.
This is used to create a network driver object.

Import json: this line imports the json module from the python standard library which is use to convert python objects into JSON format and vice versa.

Driver: get_network_driver('ios')--> this line creates a network driver object for cisco ios devices by using net_network_driver() function in NAPALM.

.

8 Sep 2024 Session 16:

Example-3:

We need to obtain facts of the devices:

get_facts : It is a method which includes device details such as hostname, model, serial number, vendor and OS version.

==========

```
from napalm import get_network_driver
import json

Driver = get_network_driver('ios')

device_details = {
    "hostname":"10.1.1.1",
    "username":"cisco",
    "password":"123456789"
}

Device_Connection = Driver(**device_details)
```

```
Device_Connection.open()

print(f"Connecting to {device_details['hostname']}")

### Get Facts  Method
Facts = Device_Connection.get_facts()
#print(Facts)

#The json.dumps() Method is used to convert Python Object into JSON
format.
Output_json = json.dumps(Facts,indent=5)
#print(Output_json)

#The json.loads() Method is used to Parse a JSON string and Convert it
into a Python Object.
Parse_string= json.loads(Output_json)
print(Parse_string)

print(f"Closing The Connection for Device {device_details['hostname']}")
Device_Connection.close()
```

==========

Example-4:

get_mac_address_table(): This will help you to get a mac address and if we need structured data then we can use the tabulate module.

Note: tabulate is not a python standard library so we need to install it on our PC.

pip install tabulate

```
from napalm import get_network_driver
from tabulate import tabulate

Driver = get_network_driver('ios')

device_details = {
    "hostname":"10.1.1.1",
    "username":"cisco",
    "password":"123456789"
}

Device_Connection = Driver(**device_details)

Device_Connection.open()

print(f"Connecting to {device_details['hostname']}")

Mac_Output= Device_Connection.get_mac_address_table()
print(Mac_Output)

table = []

for entry in Mac_Output:
    table.append([entry["interface"],entry["mac"],entry["vlan"]])
print(tabulate(table,headers=["INTERFACE",'MAC
ADDRESS',"VLAN"],tablefmt='orgtbl'))
print(f"Closing The Connection for Device {device_details['hostname']}")
Device_Connection.close()
```

Example-5:

Get_arp_table Method: This method helps to get the ARP details of the device in the table.

```
from napalm import get_network_driver
from tabulate import tabulate

Driver = get_network_driver('ios')

device_details = {
    "hostname":"10.1.1.1",
    "username":"cisco",
    "password":"123456789"
}

Device_Connection = Driver(**device_details)

Device_Connection.open()
print(f"Connecting to {device_details['hostname']}")

### Get arp table Method
ARP_Table = Device_Connection.get_arp_table()
#print(Facts)

print(f"Closing The Connection for Device {device_details['hostname']}")
Device_Connection.close()

table = []

for entry in ARP_Table:

table.append([entry["interface"],entry["age"],entry["ip"],entry["mac"]])
print(tabulate(table,headers=["INTERFACE",'AGE',"IP","MAC
ADDRESS"],tablefmt='orgtbl'))
print(f"Closing The Connection for Device {device_details['hostname']}")
Device_Connection.close()
```

Example-6:
Before using this script we need to change in our GNS3 device.

Got to device config→ memories and disk → give 30 MIB in PCMCIA disk0

Config this command to device:

ip scp server enable

R1(config)#ip scp server enable
R1(config)#

```
from napalm import get_network_driver

Driver = get_network_driver('ios')

Device_Connection =
Driver(hostname='10.1.1.1',username='cisco',password='123456789')

Device_Connection.open()
```

```
print(f"Connection Successfully")

Device_Connection.load_merge_candidate(config="interface loopback30\n ip
address 10.10.200.1 255.255.255.0")

print(Device_Connection.compare_config())

if len(Device_Connection.compare_config())>0:
    choice= input("\n Would You like to Commit thses changes?[y/N]:")
    if choice =="y":
        print("Committing...")
        Device_Connection.commit_config()
        choice= input("\n Would You like to Rollback to previous
Config?[y/N]:")
        if choice =="y":
            print("Rolling back to previous Config..")
            Device_Connection.rollback()
        else:
            print("Discarding..")
            Device_Connection.discard_config()
    else:
        print("No Difference")

print(f"Closing The Connection")
Device_Connection.close()
```

---

21 Sep 2024 Session 17:

Changing Device Config, Replace,Merge and Commit from file

Example-7:

Create a one IP_Route.txt file for below config in the same folder where the .py file exists.

ip route 10.1.100.0 255.255.255.0 10.100.100.1
ip route 10.1.101.0 255.255.255.0 10.100.100.1
ip route 10.1.102.0 255.255.255.0 10.100.100.1
ip route 10.1.103.0 255.255.255.0 10.100.100.1

And save the file

```
from napalm import get_network_driver

driver = get_network_driver('ios')

switch_01 = {
    "hostname": '10.1.1.1',
    "username": "cisco",
    "password": "123456789",
    "optional_args": {"secret": "123456789"}
    }

device = driver(**switch_01)
device.open()
print("Connected successfully")

#device.load_merge_candidate(config='interface loopback10\n ip address
192.168.100.1 255.255.255.0')
device.load_merge_candidate(filename="IP_Route.txt")


print(device.compare_config())
if len(device.compare_config()) > 0:
    choice = input("\nWould you like to commit these changes? [yN]: ")
    if choice == 'y':
        print("Committing ....")
        device.commit_config()
        choice = input("\nWould you like to Rollback to previous config?
[yN]: ")
        if choice == 'y':
            print("Rolling back to previous config ...")
```

```
            device.rollback()
    else:
        print("Discarding ...")
        device.discard_config()
else:
    print("No difference")
device.close()
print("Disconnected from the device")
```

Ex: 8

Write a script to take backup of devices

```
from napalm import get_network_driver


driver = get_network_driver('ios')
switch_01 = {
    "hostname": '10.1.1.1',
    "username": "cisco",
    "password": "123456789",
    "optional_args": {"secret": "123456789"}
    }

switch_01_conn = driver(**switch_01)
switch_01_conn.open()
print(f"Connecting to {switch_01['hostname']}")
output = switch_01_conn.get_config(retrieve="running")
print(output)
Run_Config= output['running']
print(Run_Config)

with open('bkp.txt','w') as backup:
    backup.write(Run_Config)
switch_01_conn.close()
print("Disconnected from the devices!!!!")
```

Now Lets Restore The Old Config.

Ex:9

```
from napalm import get_network_driver

driver = get_network_driver('ios')

switch_01 = {
    "hostname": '10.1.1.1',
    "username": "cisco",
    "password": "123456789",
    "optional_args": {"secret": "123456789"}
    }

device = driver(**switch_01)
device.open()
print("Connected successfully")

#device.load_merge_candidate(config='interface loopback10\n ip address
192.168.100.1 255.255.255.0')
device.load_merge_candidate(filename="bkp.txt")


print(device.compare_config())
if len(device.compare_config()) > 0:
    choice = input("\nWould you like to commit these changes? [yN]: ")
    if choice == 'y':
        print("Committing ....")
        device.commit_config()
    else:
        print("Discarding ...")
        device.discard_config()
else:
    print("No difference")
device.close()
print("Disconnected from the device")
```

---

22 Sep 2024 Session 18:

Config below command to devices:

archive
 path disk0:myconfig

```
from napalm import get_network_driver

driver = get_network_driver('ios')

switch_01 = {
    "hostname": '10.1.1.1',
    "username": "cisco",
    "password": "123456789",
    "optional_args": {"secret": "123456789"}
    }

device = driver(**switch_01)
device.open()
print("Connected successfully")

#device.load_merge_candidate(config='interface loopback10\n ip address
192.168.100.1 255.255.255.0')
device.load_merge_candidate(filename="old_bkp.txt")


print(device.compare_config())
if len(device.compare_config()) > 0:
    choice = input("\nWould you like to commit these changes? [yN]: ")
    if choice == 'y':
        print("Committing ....")
        device.commit_config(revert_in=60)
    else:
        print("Discarding ...")
        device.discard_config()
else:
```

```
    print("No difference")
device.close()
print("Disconnected from the device")
```

Network Based APIs:

What is API: Application programming interface. It is an interface that enables different applications to talk to each other using a set of mechanisms and protocols.

Similarly Network API that enables communication between the network, applications , web browsers and database.

APIs that use REST architecture(representational state transfer) is also called as restful API

These API commonly use in networking.

Restful APIs use HTTP methods to gather and manipulate data.
HTTP restful API to interact with data.

HTTP Methods:

Get: Get is used to request data from a specific resource (server)

POST: POST is used to send data to the server.
PUT: PUT is used to send data to the server.

Head: Head is almost identical to get, but without the response body.

DELETE: The delete method deletes the specific resources.

Patch: Used to apply partial modification to a resource.

Common use method: get and POST

—----------

With Network Example:

1) POST:

Action: To Configure Network devices remotely.
Use Case: Add a VLAN.

2) GET:
Action: List network devices through telemetry.

Use case: List network devices through telemetry.

3) PUT/PATCH:
   Action: Modify a network config
   Use case: change a vlan name

4) DELETE:
   Action: Delete a unused VLANs
   Use case: Delete a VLAN
   API Example:

      1) RESTCONF

Definition: This is HTTP base protocol that provides a programmatic interface to access the data.

2) NETCONF and YANG API

Definition: This is MGMT network protocol that is used to manage network devices.

Introduction of YANG data Model:

YANG(Yet another next Generation):this is a data model language specifically designed for network config and state data.

It provides a structured way to define the data structures that are used by network devices.

It is divided into two major categories.

1) **Industry standard**
2) **Vendor Specific**

Industry standard models:can be defined by standard defining organizations such as IETF and openconfig.

Vendor Specific models are commonly referred to a native model since they are native only to that vendor network OS.

The goal of SDOs such as IETF and openconfig is to work toward a vendor neutral YANG model.

Yang models can be found on github:
https://github.com/yangmodels/yang

Yang models are defined hierarchically as a tree. Each element in the model is called NODE.

Each node has a name and Value or name and child nodes.

Yang is structured into modules, In general we can say modules consist of 3 main elements.

1) **Module header statements**: define a name of the modules.
2) **Revision Statements**: simply gives the history about the modules
3) **Definition statements**: This is the main body that defines the data model.

Yang is designed to be easy for humans to read and write while being machine friendly.

Structure of YANG data model: Yang data model is structured to represent network configuration in a hierarchical and modular manner.

This structure includes several key components.

Modules and SubModules:

Modules: this is primary building blocks in YANG.
It contains definitions of the data nodes, types and Configuration elements.

SubModules:Used to split a large modulus into sub modules.

DATA NOdes:

Containers: Group related nodes.
Lists: Ordered and unordered connection of entries.

28 Sep 2024 Session 19:

Compare with Python:

Yang	Python	Description
Module	Module	a complete Yang file
DATA type	DATA type	IN Python LIst, String, integers,etc. In Yang we can define type
DATA Structures		
Leaf	Variable	a single variable which can hold a single value
Leaf-List	List	a collection of leafs of the same data type
List	Dictionary	a collection of key value pairs
Container	Class	

Problem: Currently the main method of configuring the device had been via CLI or in same cases SNMP.but these methods presented a number of issues .

CLI disadvantage:

1) **CLI commands differs from vendor to vendor**

2) **CLI OUTPUT from each vendor differs and requiring separate parsing logic/Techniques for each vendor**

SNMP disadvantage:

1) **Unreliable as it inherently use UDP as its transport protocol.**
2) **Traditionally insured. Though SNMPv3 looks to address this but still comes with its own security challenges.**
3) **Lacks Standard MIB for configuring networks. That is why vendors have developed various proprietary MIB which are a barrier to managing cross vendor platforms.**

Solutions come from IETF in the form of NetCONF and YANG.

YANG: It provide a language to describe desired config(state)
Netconf: provides the protocol to deliver and perform the required operation.

Introduction to NetCONF:

YANG-> It is a data model representation of the network devices and it needs a transport mechanism to push the config and read the network device config.

Netconf(Network configuration protocol)--> it provides a transport mechanism with YANG to manage the network devices.
It is API.

Netconf uses a transport protocol.

All the vendors adopted netconf as services or agents on their platform.

We need to enable netconf on the devices.then it allows for client/server communication.

Netconf operation are perform via the RPC(Remote procedure Calls) Between client and server(Network Devices).

These RPC are sent over a secure and connection oriented session and this session has come to use SSH.

Default listening port on the netconf enabled devices is port number 830.

Each RPC made from the client request is typically encoded in XML(Extensible Markup Language) ; each reply from the server(network devices) is also typically encoded in XML format.

We can say that Netconf(Network Configuration) is a protocol defined by the IETF(Internet Engineering Task Force) to Install, Manipulate and Delete the Configuration of the Network devices.

NETCONF protocol Stack: This is also Called Netconf protocol Layers.

This stack establishes the logical connection between the network admin or network administration application and the network devices.

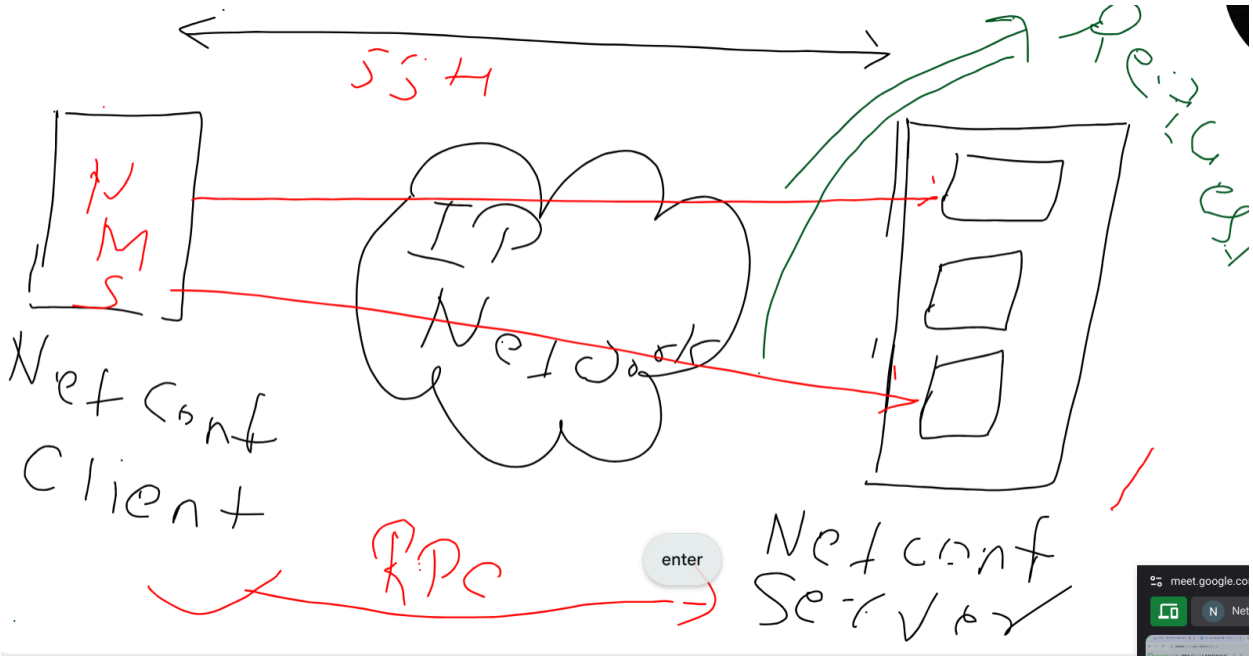
Divided into 4 Layer.

1) Transport
2) Messages
3) Operation
4) Content

1) **Transport**: This layer used to provide a communication path between the client/server (manager/agent)
2) **Message** : A set of RPC message and notification are defines for use including <rpc>,<rpc-reply>,<rep-error>.
3) **Operation**: A set of base protocol operations initiated via RPC methods using XML encoding. <get-config>,<edit-config> and <get>

4) **Content**: Netconf data model and protocol operation use the YANG modeling language. YANG represents the DATA.

How does Netconf Works:



Communication:

Netconf is based upon a Client/ Server model or Manager and Agent.

Within the Communication flow of the Netconf session there are 3 main parts.

Session Establishment: Each side sends a <hello> along with its capabilities.

Operation Request: The Client then sends its request(Operation) to the server(Network devices) via the RPC.the response is then sent back to the client within <rpc-reply>.

Session Close: The session is then closed by the Client via <close-session>.

NetConf Operation:

Opeartion	Description
<get>	Fetch Running Config and State Information
<get-config>	Fetch All or part of specified

	config data store
<edit-config>	Load all or part of a config to the specified config data store
<copy-config>	Replace the whole Config data store with another data store
<delete-config>	Delete a configuration data store
<discard-changes>	Clear all changes
<commit>	Copy candidate data store to running data store
<cancel-commit>	Cancels an ongoing Confirm Commit
<lock>	Lock the entire configuration data store so only my session can write

<close-session>	Termination of the netconf session

NCCLIENT for NETCONF:

RFC: https://datatracker.ietf.org/doc/html/rfc6241

Ncclinet is python library for netconf client

TO Install:

C:\Users\Admin>pip install ncclinet

To check it is install or not:

C:\Users\Admin>pip list

29 Sep 2024 Session 20:

Before connecting to devices with script we have to enable NetConf on the device.
To config in the devices use below command:

netconf-yang

—----------
CSR_Router#sh run | in netc

netconf-yang
CSR_Router#

—----------

Example: Let's write the first script to get the capability of the devices.

Capability: it means what model and YANG operation are supported by the device.

```
from ncclient import manager


Cisco_CSR= {
    "host":"10.1.1.5",
    "port":830,
    "username":"cisco",
    "password":"123456789",
    "device_params":{"name":"csr"}
}


rtr_mgr= manager.connect(**Cisco_CSR)

for rtr_capabilities in rtr_mgr.server_capabilities:
    print(rtr_capabilities)
```

—----------

Example-2: Use get config if we need running config:

```
from ncclient import manager

Cisco_CSR = {'host': '10.1.1.5',
           'port': 830,
           'username': 'cisco',
           'password': '123456789',
           'hostkey_verify': False,
```

```
           'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR)
output = rtr_mgr.get_config("running").data_xml
print(output)
print(type(output))
with open('conf.xml', 'w') as xml_data:
    xml_data.write(output)

rtr_mgr.close_session()
```

_——

Ex-4: Use filter and get specific config from xml:

```
from ncclient import manager

Cisco_CSR = {'host': '10.1.1.5',
           'port': 830,
           'username': 'cisco',
           'password': '123456789',
           'hostkey_verify': False,
           'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR)
hostname_filter = '''
    <filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <hostname/>
      </native>
    </filter>
'''

output = rtr_mgr.get_config('running', hostname_filter)
print(output)

rtr_mgr.close_session()
```

Filter: this is outermost element that defines the entire filter expression. It uses the namespace

```
Urn:ietf:params:xml:ns:netconf:base:1.0: this indicates that it adheres to
the netconf configuration data model.
```

——--

Ex: Config Hostname using ncclinet:

```
from ncclient import manager

Cisco_CSR = {'host': '10.1.1.5',
           'port': 830,
           'username': 'cisco',
           'password': '123456789',
           'hostkey_verify': False,
           'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR)
hostname_config = '''
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <hostname>WAN_ROUTER</hostname>
      </native>
    </config>
'''

output = rtr_mgr.edit_config(hostname_config, target= "running")
print(output)

rtr_mgr.close_session()
```

——--

Ex: Config Interface using ncclient:

```
from ncclient import manager

Cisco_CSR = {'host': '10.1.1.5',
           'port': 830,
           'username': 'cisco',
           'password': '123456789',
           'hostkey_verify': False,
           'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR)
interface_config = '''
    <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <interface>
         <GigabitEthernet>
         <name>2</name>
         <ip>
          <address>
           <primary>
            <address>192.168.1.1</address>
            <mask>255.255.255.0</mask>
           </primary>
          </address>
         </ip>
         </GigabitEthernet>
        </interface>
      </native>
    </config>
'''

output = rtr_mgr.edit_config(interface_config, target= "running")
print(output)

rtr_mgr.close_session()
```

Ex: Save Config:

```
from ncclient import manager, xml_

Cisco_CSR = {'host': '10.1.1.5',
             'port': 830,
             'username': 'cisco',
             'password': '123456789',
             'hostkey_verify': False,
             'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR)
save = """<cisco-ia:save-config
xmlns:cisco-ia="http://cisco.com/yang/cisco-ia"/>"""
output = rtr_mgr.dispatch(xml_.to_ele(save))
print(output)

rtr_mgr.close_session()
```

EX:

Get save output in Pretty xml format:

```
from ncclient import manager
import xml.dom.minidom as md

Cisco_CSR = {'host': '10.1.1.5',
             'port': 830,
             'username': 'cisco',
             'password': '123456789',
             'hostkey_verify': False,
             'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR)
output = rtr_mgr.get_config('running')
xmldata = md.parseString(str(output))
new_data = xmldata.toprettyxml(indent="  ")
print(new_data)
with open('new_conf.xml', 'w') as data:
```

```
    data.write(new_data)

rtr_mgr.close_session()
```

Ex:
Get Interface output in Pretty xml format:

```
from ncclient import manager
import xml.dom.minidom as md

Cisco_CSR = {'host': '10.1.1.5',
           'port': 830,
           'username': 'cisco',
           'password': '123456789',
           'hostkey_verify': False,
           'device_params': {'name': 'csr'}}

rtr_mgr = manager.connect(**Cisco_CSR) # To connect CSR router
interface_filter = '''
    <filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
      <native xmlns="http://cisco.com/ns/yang/Cisco-IOS-XE-native">
        <interface/>
      </native>
    </filter>
'''

output = rtr_mgr.get_config('running', interface_filter)
#print(output)
xmldata = md.parseString(str(output))
#print(xmldata)
new_data = xmldata.toprettyxml(indent="  ")
print(new_data)
with open('all_int.xml', 'w') as data:
     data.write(new_data)

rtr_mgr.close_session()
```

—----------

RESTConf:

Definition: **RestConf is HTTP base protocol that provides a restful programing interface and it allows users to delete, add, modify as well as query on network devices data.**

Restconf is developed based on the both NETCONF and HTTP and it provides core NETCONF functions using HTTP method:

RestCONF Network Architecture:

Main element in the RestConf is:

1) **RestConf Client:**

Client uses RESTCONF to manage network devices and it sends a request to the server(Network Devices) to create, delete, modify or query one or more data records.

2) **RestConf Server: To maintain the data of the managed network devices.**
   It responds to client requests and reports requested data to clients.

RestConf client and server run HTTP to establish a secure and connection oriented session.

The Request and Respond msg are coded in either XML or JSON(javascript object Notation) format.

RestConf also uses YANG as its modeling language.

Yang is a data model language used to model both the configuration and state data in Restconf.

—----------

5 Oct 2024 Session 21:

Restconf operation methods:

Config RESTCONF on the Device:

Setting up RESTCONG:

#Config t
#Restconf
#ip http secure-server
#ip http authentication local

TO Verify RESTCONG and HTTP enable or not:

#show platform software yang-management process.

Output:

```
Cisco_New#show platform software yang-management process
confd            : Running
nesd             : Running
syncfd           : Running
ncsshd           : Running
dmiauthd         : Running
nginx            : Running
ndbmand          : Running
pubd             : Running
```

To Download Postman:

https://www.postman.com/downloads/

If we have download then we need to to some setup:

Ex: Use postman to send GET Request:

Standard RESTCONF RFC: https://datatracker.ietf.org/doc/html/rfc8040

Vendor specific:
https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/171/b_171_programmability_cg/restconf_protocol.html

GET REQUEST: https://10.1.1.5/.well-known/host-meta

Response:

```
<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
    <Link rel='restconf' href='/restconf'/>
</XRD>
```

Ex:-2: Send get Request:

Request: **https://10.1.1.5/restconf/**

Output:

```
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
    <data/>
    <operations/>
    <yang-library-version>2016-06-21</yang-library-version>
</restconf>
```

Ex-3: Retrieve the Devices Module Information:

Request:
https://10.1.1.5/restconf/data/ietf-yang-library:modules-state


```
<modules-state xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library"
xmlns:yanglib="urn:ietf:params:xml:ns:yang:ietf-yang-library">
    <module-set-id>cb5b464e55b03bca514ea6537e090e2d</module-set-id>
    <module>
        <name>ATM-FORUM-TC-MIB</name>
        <revision></revision>
        <schema>https://10.1.1.5:443/restconf/tailf/modules/ATM-FORUM-TC-MIB</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:ATM-FORUM-TC-MIB</namespace>
        <conformance-type>import</conformance-type>
    </module>
    <module>
        <name>ATM-MIB</name>
        <revision>1998-10-19</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/ATM-MIB/1998-10-19</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:ATM-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>ATM-TC-MIB</name>
        <revision>1998-10-19</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/ATM-TC-MIB/1998-10-19</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:ATM-TC-MIB</namespace>
        <conformance-type>import</conformance-type>
    </module>
    <module>
        <name>BGP4-MIB</name>
        <revision>1994-05-05</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/BGP4-MIB/1994-05-05</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:BGP4-MIB</namespace>
        <conformance-type>implement</conformance-type>
```

```
    </module>
    <module>
        <name>BRIDGE-MIB</name>
        <revision>2005-09-19</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/BRIDGE-MIB/2005-09-19</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:BRIDGE-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-AAA-SERVER-MIB</name>
        <revision>2003-11-17</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-AAA-SERVER-MIB/2003-11-17</s
chema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-AAA-SERVER-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-AAA-SESSION-MIB</name>
        <revision>2006-03-21</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-AAA-SESSION-MIB/2006-03-21</
schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-AAA-SESSION-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-AAL5-MIB</name>
        <revision>2003-09-22</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-AAL5-MIB/2003-09-22</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-AAL5-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-ATM-EXT-MIB</name>
        <revision>2003-01-06</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-ATM-EXT-MIB/2003-01-06</sche
ma>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-ATM-EXT-MIB</namespace>
```

```
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-ATM-PVCTRAP-EXTN-MIB</name>
        <revision>2003-01-20</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-ATM-PVCTRAP-EXTN-MIB/2003-01
-20</schema>

<namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-ATM-PVCTRAP-EXTN-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-ATM-QOS-MIB</name>
        <revision>2002-06-10</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-ATM-QOS-MIB/2002-06-10</sche
ma>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-ATM-QOS-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-BGP-POLICY-ACCOUNTING-MIB</name>
        <revision>2002-07-26</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-BGP-POLICY-ACCOUNTING-MIB/20
02-07-26</schema>

<namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-BGP-POLICY-ACCOUNTING-MIB</namespac
e>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-BGP4-MIB</name>
        <revision>2010-09-30</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-BGP4-MIB/2010-09-30</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-BGP4-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-BULK-FILE-MIB</name>
```

```
        <revision>2002-06-10</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-BULK-FILE-MIB/2002-06-10</sc
hema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-BULK-FILE-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-CBP-TARGET-MIB</name>
        <revision>2006-05-24</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CBP-TARGET-MIB/2006-05-24</s
chema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CBP-TARGET-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-CBP-TARGET-TC-MIB</name>
        <revision>2006-03-24</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CBP-TARGET-TC-MIB/2006-03-24
</schema>

<namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CBP-TARGET-TC-MIB</namespace>
        <conformance-type>import</conformance-type>
    </module>
    <module>
        <name>CISCO-CBP-TC-MIB</name>
        <revision>2008-06-24</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CBP-TC-MIB/2008-06-24</schem
a>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CBP-TC-MIB</namespace>
        <conformance-type>import</conformance-type>
    </module>
    <module>
        <name>CISCO-CDP-MIB</name>
        <revision>2005-03-21</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CDP-MIB/2005-03-21</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CDP-MIB</namespace>
        <conformance-type>implement</conformance-type>
```

```
    </module>
    <module>
        <name>CISCO-CEF-MIB</name>
        <revision>2006-01-30</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CEF-MIB/2006-01-30</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CEF-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-CEF-TC</name>
        <revision>2005-09-30</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CEF-TC/2005-09-30</schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CEF-TC</namespace>
        <conformance-type>import</conformance-type>
    </module>
    <module>
        <name>CISCO-CONFIG-COPY-MIB</name>
        <revision>2005-04-06</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CONFIG-COPY-MIB/2005-04-06</
schema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CONFIG-COPY-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-CONFIG-MAN-MIB</name>
        <revision>2007-04-27</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CONFIG-MAN-MIB/2007-04-27</s
chema>
        <namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CONFIG-MAN-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-CONTEXT-MAPPING-MIB</name>
        <revision>2008-11-22</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-CONTEXT-MAPPING-MIB/2008-11-
22</schema>
```

```
<namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-CONTEXT-MAPPING-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
    <module>
        <name>CISCO-DATA-COLLECTION-MIB</name>
        <revision>2002-10-30</revision>

<schema>https://10.1.1.5:443/restconf/tailf/modules/CISCO-DATA-COLLECTION-MIB/2002-10-30</schema>

<namespace>urn:ietf:params:xml:ns:yang:smiv2:CISCO-DATA-COLLECTION-MIB</namespace>
        <conformance-type>implement</conformance-type>
    </module>
```

Ex-4: Use a get request t gather the information for all the interface in JSON:

Go to header and in key add accept and in value field type add application/yang-data+json

https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/171/b_171_programmability_cg/restconf_protocol.html

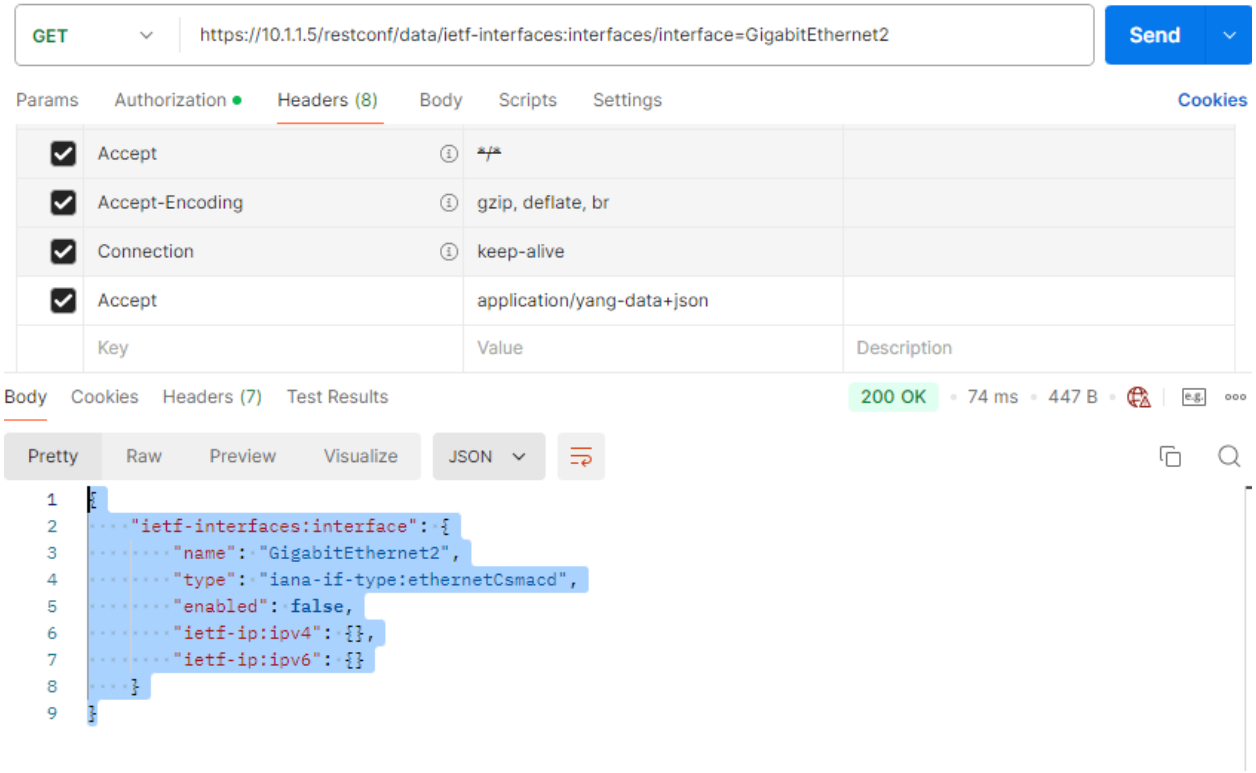
Ex: Fetch information for specific interface:

https://10.1.1.5/restconf/data/ietf-interfaces:interfaces/interface=GigabitEthernet2

Output:


```
{
    "ietf-interfaces:interface": {
```

```
        "name": "GigabitEthernet2",
        "type": "iana-if-type:ethernetCsmacd",
        "enabled": false,
        "ietf-ip:ipv4": {},
        "ietf-ip:ipv6": {}
    }
}
```



Ex: To Config Loopback interface with PUT method:

Go to body and click on raw and then update your config in DATA:

```
{
  "ietf-interfaces:interface": {
    "name": "Loopback1",
    "description": "This_Is_Loopback_Interface",
    "type": "iana-if-type:softwareLoopback",
    "enabled": true,
    "ietf-ip:ipv4": {
      "address": [
```

```
        {
          "ip": "10.1.10.1",
          "netmask": "255.255.255.0"
        }
      ]
    },
    "ietf-ip:ipv6": {}
  }
}
```


Request:
https://10.1.1.5/restconf/data/ietf-interfaces:interfaces/interface=Loopback1

Response: 201 Created

6 Oct 2024 Session 22:

Ansible:

1. Ansible is an open source IT engine which automates application deployment.
2. It used to monitor, configure and troubleshoot devices in large network.
3. It is written in Python.
4. It uses SSH to Connect the devices.

5. After connecting the devices/node, ansible pushes one small program called “Ansible Modules”.
6. Ansible runs these modules on our devices and removes them when the job is done.

7. Ansible is agentless.
8. It use PUSH Mechanism
9. It use YAML (yet another markup language)

YAML:

1. YAML files start with a triple Dash
2. There are two type of data format in the YAMLfile
   1)LIst
   2) Key-Value pair
3. List items are designated by a -(dash)
4. Key- value pair are designated by a Key:Value
5. Correct indentation should be followed

Note: There Should be space Between : and value

Example:

```
--- # OPtional YAML start syntax
Mohan:
    name: Mohan
    roll num: 34
    div: B
... # Optional YAML end syntax
```

```
Or We can write in this format.
Mohan:{name:" Mohan",roll num :34,div: B}
```

Example-2: Representing a LIst

```
---
   VLAN:
      - VLAN-10
      - VLAN-20
      - VLAN-30
      - VLAN-40
...

Or We can write in this way

VLAN:["VLAN-10","VLAN-20","VLAN-30","VLAN-40"]
```

Playbook:

Playbook are the files where ansible code is written
Playbooks are written in YAML format.
It contain a list of task
Playbook are run sequentially

Ex:

```
---
- name: Cisco Show verion example
  hosts: routes
  gather_facts: false

  tasks:
   - name: show show on the router
```

```
    ios_command:
      commands: show version | in Version
    register: output
  - name: Print Output
    debug:
       var: output.stdout_lines
```

Explain:
Name: Any arbitrary name
Hosts: Referring to the inventory group called routers
Gather_facts: We don't need to gather information from the router. This will be useful when working with linux server.

Task: a task is a section which consist of single procedure to complete.

Register: we can create variables from the output of the ansible task with task keyword register.

We can use registered variables in any task in our play.

Debug: this module's print statement during execution.

Stdout_lines: print the output in readable format.

—----------

Ansible Server: it is a machine where ansible is installed and from which all tasks and playbooks will be executed.

Modules: The module is a command or set of similar commands which is executed on the client side.

Task: a task is a section which consists of a single procedure to complete.

Role: It is a way of organizing tasks.

Inventory: It contain all devices information

Play: It is execution of playbook

—----------

To Install Ansible:

ajayyadav941@DESKTOP-G42GSMH:~$ sudo su
[sudo] password for ajayyadav941:
root@DESKTOP-G42GSMH:/home/ajayyadav941#
sudo apt update

#sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible

To Check it Is inStalL or not

```
root@DESKTOP-G42GSMH:/home/ajayyadav941# ansible --version
ansible [core 2.12.10]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.8.10 (default, Mar 13 2023, 10:26:41) [GCC 9.4.0]
  jinja version = 2.10.1
  libyaml = True
root@DESKTOP-G42GSMH:/home/ajayyadav941#
root@DESKTOP-G42GSMH:/home/ajayyadav941#
```

12 Oct 2024 Session 23:

Ansible DOC:
https://docs.ansible.com/ansible/latest/collections/cisco/ios/index.html

https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html
Inventory file:

We need t update below details in this file.

#This is My Routers for Testing
[routers]
router-1 ansible_host=10.1.1.1
router-2 ansible_host=10.1.1.2

[routers:vars]
ansible_network_os=ios
ansible_user=cisco
ansible_password=123456789
ansible_connection=network_cli

Ansible_network_os: To inform ansible which network platform this host corresponds to.

ansible_user= The user to connect the other remote device.

ansible_password=Password for the ansible_user.
ansible_connection=When working with Network devices we need to set this always network_cli.

Ansible_become: if enable mode should be used.

ansible-playbook show_version.yml -i /etc/ansible/hosts

Example: Fetch Show version of devices.

```
---
- name: Cisco show version example
  hosts: routers
  gather_facts: false   #For Network Devices we need to use false always
  tasks:
    - name: run show version on the routers
      ios_command:
```

```
      commands: show version | incl Version
    register: output
  - name: print output
    debug:
      var: output.stdout_lines
```

If we need to run this play use below command:

ansible-playbook show_version.yml -i /etc/ansible/hosts

—----------

Output:

root@DESKTOP-G42GSMH:/etc/ansible/playbooks# ansible-playbook show_version.yml -i /etc/ansible/hosts

PLAY [Cisco show version example] **********

****

****

TASK [run show version on the routers] **********

****

***

```
fatal: [router-3]: FAILED! => {"changed": false, "msg":
"[Errno -2] Name or service not known"}
ok: [router-2]
ok: [router-1]

TASK [print output]
*****************************************************************
*****************************************************************
**********************
ok: [router-2] => {
    "output.stdout_lines": [
        [
            "Cisco IOS Software, 7200 Software
(C7200-ADVENTERPRISEK9-M), Version
15.3(3)XB12, RELEASE SOFTWARE (fc2)",
            "BOOTLDR: 7200 Software
(C7200-ADVENTERPRISEK9-M), Version
15.3(3)XB12, RELEASE SOFTWARE (fc2)",
            "6 slot VXR midplane, Version 2.1"
        ]
    ]
}
ok: [router-1] => {
    "output.stdout_lines": [
        [
```

```
            "Cisco IOS Software, 7200 Software
(C7200-ADVENTERPRISEK9-M), Version
15.3(3)XB12, RELEASE SOFTWARE (fc2)",
            "BOOTLDR: 7200 Software
(C7200-ADVENTERPRISEK9-M), Version
15.3(3)XB12, RELEASE SOFTWARE (fc2)",
            "6 slot VXR midplane, Version 2.1"
        ]
    ]
}

PLAY RECAP
*********************************************************************
*********************************************************************
*********************************
router-1                   : ok=2    changed=0
unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
router-2                   : ok=2    changed=0
unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
router-3                   : ok=0    changed=0
unreachable=0    failed=1    skipped=0    rescued=0
ignored=0
```

root@DESKTOP-G42GSMH:/etc/ansible/playbooks#

—----------

Example:

root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
cat config.yml

```
---
- name: Configuring Cisco IOS Devices
  gather_facts: no
  hosts: Cisco
  connection: network_cli

  tasks:
    - name: Basic Configuration
      ios_config:
        save_when: modified
        lines:
          - ip name-server 8.8.8.8
          - ip http secure-server
      register: output

    - name: Printing at Config Command
```

```
    debug: var=output
...
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#

Output:

root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
ansible-playbook config.yml -i /etc/ansible/hosts

PLAY [Configuring Cisco IOS Devices]
**********************************************************************
**********************************************************************
*****

TASK [Basic Configuration]
**********************************************************************
**********************************************************************
***************
fatal: [router-3]: FAILED! => {"changed": false, "msg":
"[Errno -2] Name or service not known"}
[WARNING]: To ensure idempotency and correct diff
the input configuration lines should be similar to how
they appear if present in the running configuration on
device
changed: [router-2]
```

```
changed: [router-1]

TASK [Printing at Config Command]
**********************************************************************
**********************************************************************
********
ok: [router-2] => {
    "output": {
        "banners": {},
        "changed": true,
        "commands": [
            "ip name-server 8.8.8.8",
            "ip http secure-server"
        ],
        "failed": false,
        "updates": [
            "ip name-server 8.8.8.8",
            "ip http secure-server"
        ],
        "warnings": [
            "To ensure idempotency and correct diff the
input configuration lines should be similar to how they
appear if present in the running configuration on
device"
        ]
```

```
    }
}
ok: [router-1] => {
    "output": {
        "banners": {},
        "changed": true,
        "commands": [
            "ip name-server 8.8.8.8",
            "ip http secure-server"
        ],
        "failed": false,
        "updates": [
            "ip name-server 8.8.8.8",
            "ip http secure-server"
        ],
        "warnings": [
            "To ensure idempotency and correct diff the input configuration lines should be similar to how they appear if present in the running configuration on device"
        ]
    }
}
```

```
PLAY RECAP
*********************************************************************
*********************************************************************
*******************************
router-1                   : ok=2    changed=1
unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
router-2                   : ok=2    changed=1
unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
router-3                   : ok=0    changed=0
unreachable=0    failed=1    skipped=0    rescued=0
ignored=0

root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
```

—----------

Ex-3: Take backup of devices:

```
---
- name: Configuring Cisco IOS Devices
  gather_facts: no
  hosts: routers
  connection: network_cli
```

```
  tasks:
    - name: Backup running-config
      ios_config:
        backup: yes
...
```

—----------

Output:

```
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
ansible-playbook backup.yml -i /etc/ansible/hosts

PLAY [Configuring Cisco IOS Devices]
**********************************************************************
**********************************************************************
*****

TASK [Backup running-config]
**********************************************************************
**********************************************************************
*************
fatal: [router-3]: FAILED! => {"changed": false, "msg":
"[Errno -2] Name or service not known"}
changed: [router-2]
```

changed: [router-1]

PLAY RECAP
****
****
****
router-1 : ok=1 changed=1
unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
router-2 : ok=1 changed=1
unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
router-3 : ok=0 changed=0
unreachable=0 failed=1 skipped=0 rescued=0
ignored=0

—----------

12 Oct 2024 Session 24:

Git and Git Hub:

Git: Git is a version control system.
Git helps keep track of code changes.

Git is used to collaborate on the code.
Git is a local repository.

Git Install:

For Debian/Ubuntu

For the latest stable version for your release of Debian/Ubuntu

```
# apt-get install git
```

For Ubuntu, this PPA provides the latest stable upstream Git version

```
# add-apt-repository ppa:git-core/ppa # apt update; apt install git
```

https://git-scm.com/downloads/linux

For windows:

https://git-scm.com/downloads/win

To check version of version:

Command:
git –version

Output:

```
root@DESKTOP-G42GSMH:/home/ajayyadav941#
git --version
git version 2.25.1
root@DESKTOP-G42GSMH:/home/ajayyadav941#
```

—----------

To check all command:

```
Git –help
```

—----------

Output:

```
root@DESKTOP-G42GSMH:/home/ajayyadav941#
git --help
usage: git [--version] [--help] [-C <path>] [-c
<name>=<value>]
           [--exec-path[=<path>]] [--html-path]
[--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager]
[--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>]
[--namespace=<name>]
           <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone   Clone a repository into a new directory
   init   Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add   Add file contents to the index
   mv   Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm   Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect   Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc
grep Print lines matching a pattern
log Show commit logs
show Show various types of objects
status Show the working tree status

grow, mark and tweak your common history
branch List, create, or delete branches
commit Record changes to the repository
merge Join two or more development histories together
rebase Reapply commits on top of another base tip
reset Reset current HEAD to the specified state
switch Switch branches
tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
fetch Download objects and refs from another repository
pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
root@DESKTOP-G42GSMH:/home/ajayyadav941#
—----------

Git Config:

git config --global user.name "ajayyadav941"
git config --global user.email
"ajayadav941@gmail.com

—----------
root@DESKTOP-G42GSMH:/home/ajayyadav941#
git config --global user.email
"ajayadav941@gmail.com"
root@DESKTOP-G42GSMH:/home/ajayyadav941#
git config --global user.name "ajayyadav941"
root@DESKTOP-G42GSMH:/home/ajayyadav941#

—----------

If we can done git config then we need to create one directory and we have to initialize git.

Command to create directory:
root@DESKTOP-G42GSMH:/etc/ansible# mkdir Ajay

Command to initialize git:

root@DESKTOP-G42GSMH:/etc/ansible# git init
Initialized empty Git repository in /etc/ansible/.git/
root@DESKTOP-G42GSMH:/etc/ansible#

—----------

git status: To see if it is part of the local repo.

In git file found in 2 state:

1) Tracked: Files that git know about and added in repo.

2) Untracked: files that are in your working directory but not added in the local repo.

—----------

Output when we are in working directory

```
root@DESKTOP-G42GSMH:/etc/ansible/playbooks# git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   show_version.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .backup.yml.swo
        Inventory.yml
        backup.yml
        backup/
        config.yml
```

show_version_1.yml

no changes added to commit (use "git add" and/or "git commit -a")
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#

—----------

git add: add files from the current/working directory to the staging environment.

git add --all or git add -A: add all files from the current/working directory to the staging environment.

—----------

If we have added file from working to staging then if we will use again git status command, will get below output:

—----------

Output:

root@DESKTOP-G42GSMH:/etc/ansible/playbooks# git status
On branch main

Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified: show_version.yml

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .backup.yml.swo
        Inventory.yml
        backup.yml
        backup/
        config.yml
        show_version_1.yml

root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
—--

Git commit: Since we have finished our work, we are ready to move from stage to commit for our repo.

We can use the commit command with messages.

Command:

Git commit -m <give message>

Output:

—--

root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
git commit -m "This is my second changes"
[main 7389ac4] This is my second changes
1 file changed, 3 insertions(+), 3 deletions(-)
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#

—--

git log: To view the history of commits for a repo. We can use the log command.

—--

Output:

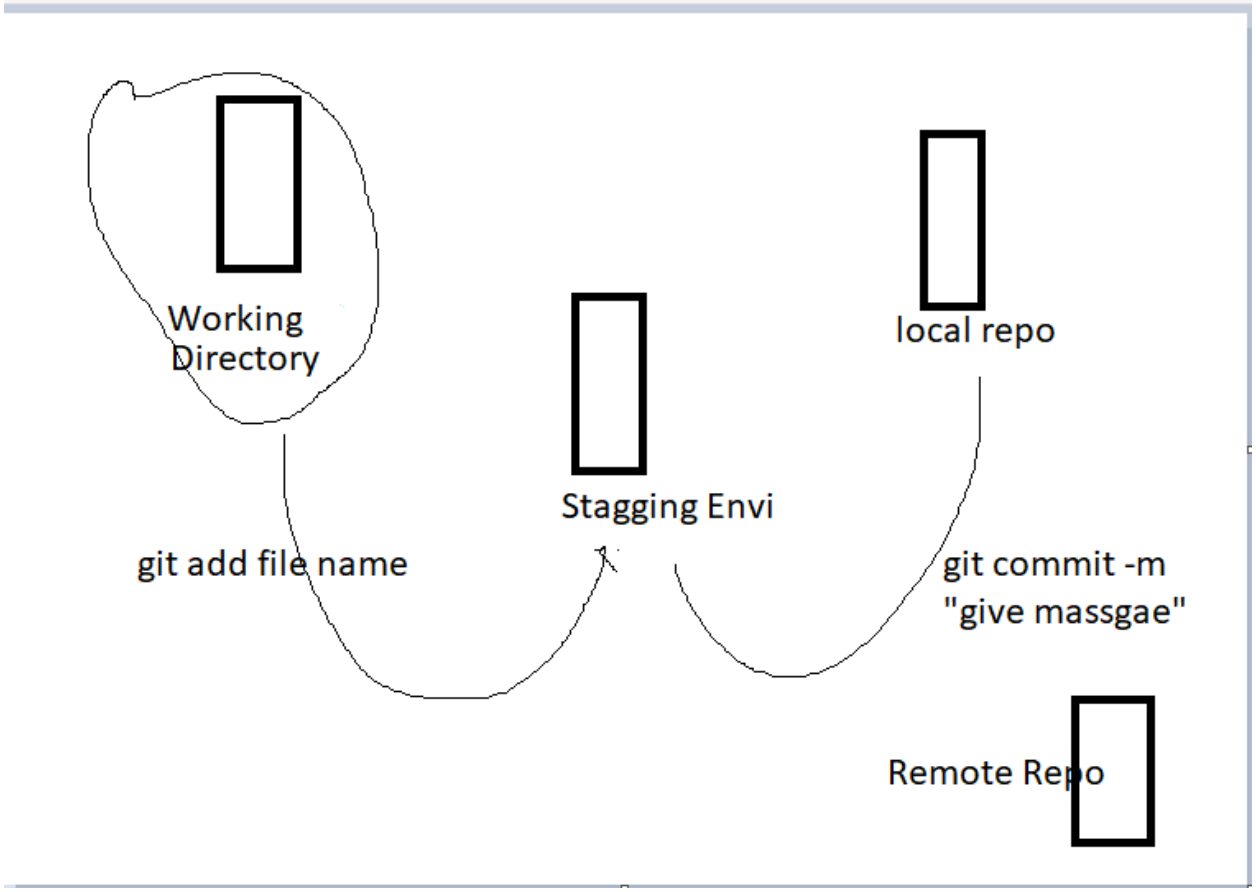
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
git log
commit
7389ac403c09c2dc17495eb802f3c9b2563141cb
(HEAD -> main)

Author: ajayyadav941 <ajayadav941@gmail.com>
Date: Sun Oct 13 19:17:07 2024 +0530

This is my second changes

commit 8f1fde7db0d05ab1694c4eda7316c544d4ed0726 (origin/main, master)
Author: ajayyadav941 <cisco>
Date: Sat Oct 5 18:57:09 2024 +0530

This is my first script
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
—----------



Git branch: In git a branch is a new/separate version of the main repo.

If we need to create new branch we can use command git branch <branch_name>

Ex:
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
git branch New_config1
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#

To check branch created or not

```
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
git branch
  New_config
* New_config1
  main
  master
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
```

—----------

To Move from branch one to other branch

Commad:

git checkout <branch_name>

—----------

Ex:

```
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
git checkout New_config1
Switched to branch 'New_config1'
root@DESKTOP-G42GSMH:/etc/ansible/playbooks#
```

—----------

GitHub: it is a remote repo.

To sync github with git

git remote add origin
https://<token_add>@github.com/ajayyadav941/ajay_new.git

To push from local to remote repo.
git push -u origin main

—----------

Paramiko:

1) It is a pure python implementation of ssh2 protocol.
2) It provides client and server functionally.
3) It provides the foundation for the high level ssh library.

To Install from CMD:

Pip install paramiko

```
import paramiko

ssh_client = paramiko.SSHClient()

ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

```
print("Connecting to the devcies 10.1.1.10")
ssh_client.connect(hostname="10.1.1.1",username="cisco",password="12345678
9",look_for_keys=False,allow_agent=False)


#Checking if the connection is active ot not
print(ssh_client.get_transport().is_active())

print("Closing the conection")

ssh_client.close()
```

Ex:

```
import paramiko
import time

ssh_client = paramiko.SSHClient()

ssh_client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

print("Connecting to the devcies 10.1.1.10")
ssh_client.connect(hostname="10.1.1.1",username="cisco",password="12345678
9",look_for_keys=False,allow_agent=False)


#Checking if the connection is active ot not
print(ssh_client.get_transport().is_active())

shell= ssh_client.invoke_shell()

shell.send("terminal len 0\n")
shell.send("Show Version\n")
shell.send("show ip int bri\n")

time.sleep(5)
```

```
#reading from the shell output buffer
Output = shell.recv(10000)
print(Output.decode("utf-8"))
print("Closing the connection")

ssh_client.close()
```

Telnetlib:

The telnetlib is a Python module, that provides a `Telnet()` class that implements the Telnet protocol. Python's telnetlib lets you easily automate access to Telnet servers (device). The telnetlib library is already included in the python package.
`telnetlib` works at the byte level, so you need to encode commands as bytes and decode responses from bytes to strings.

With Python 'help()' function we check the 'telnetlib' module doc:

>>> help('telnetlib')

Help on module telnetlib:

 # Output is Omitted

Example:

>>> from telnetlib import Telnet

```
>>> tn = Telnet('www.python.org', 79)   # connect to finger port
>>> tn.write(b'guido\r\n')
>>> print(tn.read_all())
Login       Name               TTY         Idle    When    Where
guido    Guido van Rossum      pts/2        <Dec  2 11:10>
snag.cnri.reston.
```

To Perform Connection:

telnet = telnetlib.Telnet('192.168.100.1')

Method read_until:

Method read_until specifies till which line the output should be read. However, as an argument, it is necessary to pass bytes, not the usual string:

In other words:

It allows you to read data from the server until a specific pattern is encountered.

Purpose:

1) Reads data (bytes) from the Telnet server in a controlled manner.
2) Stops reading once the specified pattern (expected value) is found in the data stream.

In [2]: telnet.read_until(b'Username')

Out[2]: b'\r\n\r\nUser Access Verification\r\n\r\nUsername'

Method write:

The write method in the telnetlib module of Python allows you to send data to a Telnet devices. You must pass a byte string as an argument:

Purpose:

1) Used to transmit information (commands, data) to the remote Telnet server.

Python Syntax:

```
tn.write(data)
```

Example: telnet.write(b'cisco\n')

Method read_very_eager:

1) This is another read method.

2)This method help to send multiple commands and then read all available output

Example:

```
telnet.write(b'sh arp\n')
telnet.write(b'sh clock\n')
telnet.write(b'sh ip int br\n')
all_result = telnet.read_very_eager().decode('utf-8')
```

Note:

You should always set time.sleep(n) before using read_very_eager.

Method expect: Method expect allows you to specify a list with regular expressions.

Method close: Method close closes connection

Script:

```
import telnetlib
import time

host = '10.1.1.10'
port = '23'
user = 'u1'
password = 'cisco'

tn = telnetlib.Telnet(host=host, port=port)

tn.read_until(b'Username: ')
tn.write(user.encode() + b'\n')

tn.read_until(b'Password: ')
tn.write(password.encode() + b'\n')
```

```
tn.write(b'terminal length 0\n')
# tn.write(b'show ip interface brief\n')
tn.write('show ip int brief\n'.encode())
tn.write(b'enable\n')
tn.write(b'cisco\n') # this is the enable password
tn.write(b'show run\n')
tn.write(b'exit\n')
time.sleep(1)

output = tn.read_all()
print(type(output))
output = output.decode()
print(output)
```

To configure the default route with multiple routers.

```
import telnetlib
import time

router1 = {'host': '10.1.1.1', 'user': 'cisco', 'password': '123456789'}
```

```
router2 = {'host': '10.1.1.2', 'user': 'cisco', 'password': '123456789'}
router3 = {'host': '10.1.1.3', 'user': 'cisco', 'password': '123456789'}

routers = [router1, router2, router3]

for router in routers:
    print(f'Connecting to {router["host"]}')

    tn = telnetlib.Telnet(host=router['host'])

    tn.read_until(b'Username: ')
    tn.write(router['user'].encode() + b'\n')

    tn.read_until(b'Password: ')
    tn.write(router['password'].encode() + b'\n')

    tn.write(b'terminal length 0\n')
    tn.write(b'enable\n')
    tn.write(b'cisco\n')  # this is the enable password

    tn.write(b'conf t\n')
    tn.write(b'ip route 0.0.0.0 0.0.0.0 e0/0 10.1.1.2\n')
    tn.write(b'end\n')
```

```
    tn.write(b'show ip route\n')
    tn.write(b'exit\n')
    time.sleep(1)

    output = tn.read_all()
    output = output.decode()
    print(output)
    print('#' * 50)
```

Secure Password:

```
import telnetlib
import time
import getpass

router1 = {'host': '10.1.1.10', 'user': 'u1'}
router2 = {'host': '10.1.1.20', 'user': 'u1'}
router3 = {'host': '10.1.1.30', 'user': 'u1'}

routers = [router1, router2, router3]

for router in routers:
```

```
print(f'Connecting to {router["host"]}')
password = getpass.getpass('Enter Password:')

tn = telnetlib.Telnet(host=router['host'])

tn.read_until(b'Username: ')
tn.write(router['user'].encode() + b'\n')

tn.read_until(b'Password: ')
tn.write(password.encode() + b'\n')

tn.write(b'terminal length 0\n')
tn.write(password.encode() + b'\n')
tn.write(b'cisco\n')  # this is the enable password

tn.write(b'conf t\n')
tn.write(b'ip route 0.0.0.0 0.0.0.0 e0/0 10.1.1.2\n')
tn.write(b'end\n')
tn.write(b'show ip route\n')
tn.write(b'exit\n')
time.sleep(1)

output = tn.read_all()
```

```
output = output.decode()
print(output)
print('#' * 50)
```