**Netmiko Lab 1: Introduction to Netmiko and Displaying output with send\_command()**

**Objectives Part 1**

* Become introduced to Netmiko.
* Install Netmiko.

**Objectives Part 2**

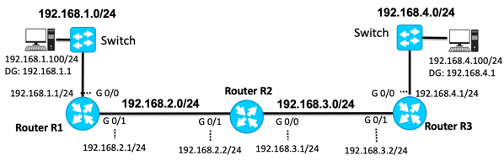
* Use Python and Netmiko to send an IOS command to a router and display the output.

**Requirements**

* Python: You should be familiar with the basics of Python.
* Using the topology below or your own topology, have access to one or more Cisco IOS devices using SSH.

**Topology**

This lab uses the following topology. You can use a different topology, just be sure to make any necessary modifications to interfaces and addresses. See Netmiko Lab 0: Using a Script for Initial Setup



**What is Netmiko?**

Netmiko is a multivendor library that simplifies the process of creating ssh connections to different network devices. Netmiko is a high-level library built on top of Paramiko. It abstracts away many of the complexities of Paramiko and provides a simple and consistent interface for working with various network devices.

The purposes of this library are the following:

* Successfully establish an SSH connection to the device.
* Simplify the execution of show commands and the retrieval of output data
* Simplify execution of configuration commands including possibly commit actions
* Do the above across a broad set of networking vendors and platforms

Netmiko uses Python to provide a simple and consistent format for interacting with network devices, allowing you to automate tasks such as device configuration, gathering information from devices, and running commands remotely. It supports a wide range of network devices, including Cisco routers, switches, and firewalls.

Here is a little more detail with some key features and benefits of using Netmiko for automating Cisco networks:

* **SSH Connectivity**:Netmiko utilizes SSH (Secure Shell) to establish connections with network devices, providing a secure and reliable means of communication.
* **Device Abstraction:** Netmiko abstracts the device-specific details, allowing you to write automation scripts that work across different Cisco devices. This saves time and effort when managing a network with various device types.
* **Configuration Management:** Netmiko enables you to automate the configuration of Cisco devices. You can push configuration changes, retrieve configurations, and compare configurations between devices.
* **Command Execution:** You can remotely execute commands on Cisco devices using Netmiko. This is useful for tasks such as retrieving device information, running diagnostic commands, and verifying device status.
* **Scripting and Automation:** Netmiko integrates well with Python, making it a powerful tool for scripting and automation. You can write Python scripts to automate repetitive tasks, perform bulk changes, or orchestrate complex network operations.
* **Community Support:** Netmiko has an active community of users who contribute to its development and provide support. This means you can find resources, examples, and solutions to common problems easily.

Overall, Netmiko is widely used and recommended for automating Cisco networks due to its simplicity, versatility, and robustness. It streamlines network operations, enhances efficiency, and enables network engineers to focus on higher-level tasks by automating routine operations.

Why the name Netmiko?

Netmiko got its name from the combination of "Net" and "miko."

1. Net: The "Net" in Netmiko represents networking. Since Netmiko is primarily designed for network automation and interacting with network devices, the "Net" portion of the name signifies its focus on networking-related tasks.
2. Miko: The term "miko" is derived from the Japanese word "巫女," which translates to "shrine maiden" or "priestess." In Japanese culture, miko are considered sacred and play a role in facilitating communication between humans and deities.

The choice of the name "Netmiko" implies that the library acts as a mediator or facilitator between the user (developer or network engineer) and the network devices. It helps establish a connection and enables communication with the network devices, similar to the role of a miko in facilitating communication between humans and deities.

The name Netmiko captures the essence of the library, combining the networking aspect ("Net") and the concept of mediation or facilitation ("miko") between the user and the network devices.

Top of Form

Regenerate response

Bottom of Form

**Part 1: Getting Started with Netmiko**

**Step 1: Install Netmiko**

Installing Netmiko may vary depending on your operating system. Here are the generic instructions:

1. Open Terminal, which you can find in the "Utilities" folder within the "Applications" folder.
2. Check if Python is installed by typing the following command and pressing Enter:

**python --version**

If Python is not installed, you can download and install it from the official Python website (https://www.python.org).

1. Install Netmiko by running the following command in Terminal:

**pip install netmiko**

**Step 2: Netmiko Documentation**

When in doubt, check the documentation 😊

<https://github.com/ktbyers/netmiko>

**Step 3: Becoming familiar with Netmiko methods**

You can skip this step if you wish.

Enter Python interactive mode and perform the following commands:

>>> **import netmiko**

>>> **dir(netmiko)**

['BaseConnection', 'CNTL\_SHIFT\_6', 'ConfigInvalidException', 'ConnLogOnly', 'ConnUnify', 'ConnectHandler', 'ConnectionException', 'FileTransfer', 'InLineTransfer', 'MIN\_PYTHON\_VER', 'NetMikoAuthenticationException', 'NetMikoTimeoutException', 'Netmiko', 'NetmikoAuthenticationException', 'NetmikoBaseException', 'NetmikoTimeoutException', 'PY\_MAJ\_VER', 'PY\_MIN\_VER', 'ReadException', 'ReadTimeout', 'SCPConn', 'SSHDetect', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_path\_\_', '\_\_spec\_\_', '\_\_version\_\_', 'a10', 'accedian', 'adtran', 'alcatel', 'allied\_telesis', 'apresia', 'arista', 'aruba', 'audiocode', 'base\_connection', 'broadcom', 'brocade', 'calix', 'cdot', 'centec', 'channel', 'check\_python\_version', 'checkpoint', 'ciena', 'cisco', 'cisco\_base\_connection', 'citrix', 'cloudgenix', 'coriant', 'dell', 'dlink', 'eltex', 'endace', 'enterasys', 'ericsson', 'exceptions', 'extreme', 'f5', 'file\_transfer', 'flexvnf', 'fortinet', 'hp', 'huawei', 'ipinfusion', 'juniper', 'keymile', 'linux', 'log', 'logging', 'mellanox', 'mikrotik', 'mrv', 'netapp', 'netgear', 'netmiko\_globals', 'no\_config', 'no\_enable', 'nokia', 'oneaccess', 'ovs', 'paloalto', 'platforms', 'pluribus', 'progress\_bar', 'quanta', 'rad', 'raisecom', 'redispatch', 'ruckus', 'ruijie', 'scp\_functions', 'scp\_handler', 'session\_log', 'sixwind', 'sophos', 'ssh\_auth', 'ssh\_autodetect', 'ssh\_dispatcher', 'supermicro', 'sys', 'terminal\_server', 'tplink', 'ubiquiti', 'utilities', 'vyos', 'watchguard', 'yamaha', 'zte', 'zyxel']

>>>

The **dir(netmiko)** command lists all the attributes, classes (methods), and functions available in the netmiko. Notice the **ConnectionHandler()** method, which is among the most commonly used. The Netmiko connection handler establishes SSH connections to network devices, sends commands, and retrieves output.

Once you create a connection with the **ConnectionHandler()** method you will have access to various methods and attributes that are available on the connection object.

The ConnectionHandler methods and attributes can be displayed using the commands:

>>>**connection = netmiko.ConnectHandler(ip='192.168.1.1', device\_type='cisco\_ios', username='admin', password='cisco')**

>>>**dir(connection)**

Note: This is an example of a connection being made, which is discussed later. This creates a **connection** object pointing to ConnectHandler.

Some of the **ConnectionHandler** methods and attributes are:

['RESPONSE\_RETURN', 'RETURN', 'TELNET\_RETURN', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_enter\_\_', '\_\_eq\_\_', '\_\_exit\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', '\_autodetect\_fs', '\_build\_ssh\_client', '\_config\_mode', '\_connect\_params\_dict', '\_enter\_shell', '\_first\_line\_handler', '\_key\_check', '\_legacy\_mode', '\_lock\_netmiko\_session', '\_modify\_connection\_params', '\_multiline\_kwargs', '\_open', '\_prompt\_handler', '\_read\_buffer', '\_return\_cli', '\_sanitize\_output', '\_send\_command\_str', '\_send\_command\_timing\_str', '\_session\_locker', '\_session\_log\_close', '\_test\_channel\_read', '\_timeout\_exceeded', '\_try\_session\_preparation', '\_unlock\_netmiko\_session', '\_use\_ssh\_config', 'allow\_agent', 'allow\_auto\_change', 'alt\_host\_keys', 'alt\_key\_file', 'ansi\_escape\_codes', 'auth\_timeout', 'banner\_timeout', 'base\_prompt', 'blocking\_timeout', 'channel', 'check\_config\_mode', 'check\_enable\_mode', 'cleanup', 'clear\_buffer', 'command\_echo\_read', 'commit', 'config\_mode', 'conn\_timeout', 'delay\_factor\_compat', 'device\_type', 'disable\_paging', 'disabled\_algorithms', 'disconnect', 'enable', 'encoding', 'establish\_connection', 'exit\_config\_mode', 'exit\_enable\_mode', 'fast\_cli', 'find\_prompt', 'global\_cmd\_verify', 'global\_delay\_factor', 'host', 'is\_alive', 'keepalive', 'key\_file', 'key\_policy', 'normalize\_cmd', 'normalize\_linefeeds', 'paramiko\_cleanup', 'passphrase', 'password', 'pkey', 'port', 'protocol', 'read\_channel', 'read\_channel\_timing', 'read\_timeout\_override', 'read\_until\_pattern', 'read\_until\_prompt', 'read\_until\_prompt\_or\_pattern', 'remote\_conn', 'remote\_conn\_pre', 'run\_ttp', 'save\_config', 'secret', 'select\_delay\_factor', 'send\_command', 'send\_command\_expect', 'send\_command\_timing', 'send\_config\_from\_file', 'send\_config\_set', 'send\_multiline', 'send\_multiline\_timing', 'serial\_login', 'serial\_settings', 'session\_log', 'session\_preparation', 'session\_timeout', 'set\_base\_prompt', 'set\_terminal\_width', 'sock', 'special\_login\_handler', 'ssh\_config\_file', 'strip\_ansi\_escape\_codes', 'strip\_backspaces', 'strip\_command', 'strip\_prompt', 'system\_host\_keys', 'telnet\_login', 'timeout', 'use\_keys', 'username', 'verbose', 'write\_channel']

Highlighted are some of the Netmiko methods (commands) we will be using in these labs.

Some of the notable members are:

* **device**: A dictionary containing information about the connected device, such as IP address, device type, and credentials.
* **disconnect**: A method to close the connection to the device.
* **send\_command**: A method used to send a command to the device and retrieve the output.
* **config\_mode**: A method to enter configuration mode on the device.
* **save\_config**: A method to save the running configuration to the startup configuration on the device.

**Part 2: Sending a Show Command and Displaying the Output**

**Step 1: Open a new Python file**

Using your Python IDE or equivalent, create a new Python file.

Note: See Step 6 if you wish to see the completed program.

**Step 2: Import Netmiko**

Enter the **import netmiko** command. This command imports the Netmiko library.

**import netmiko**

**Step 3: Establish an SSH connection - Connect.Handler()**

The Netmiko connection handler class, **Connect.Handler()**, establishes SSH connections to network devices, sends commands, and retrieves output. It manages authentication, prompts, and error handling, simplifying the process of interacting with network devices. By abstracting these details, it provides a convenient interface for automating tasks on Cisco devices.

**connection = netmiko.ConnectHandler(ip='192.168.1.1',**

**device\_type='cisco\_ios',**

**username='admin',**

**password='cisco')**

This is the information, arguments, required for the SSH connection including:

* **ip**: This parameter specifies the IP address of the device you want to connect to. In this case, it is set to **192.168.1.1**. You should replace it with the actual IP address of your target device.
* **device\_type**: This parameter specifies the type of the network device you are connecting to. In this example, it is set to '**cisco\_ios**', indicating that the device is a Cisco IOS-based device. Netmiko supports various device types from different vendors, and the appropriate device type should be chosen based on the target device you are working with.
* **username** and **password**: These parameters represent the credentials required to authenticate and establish the connection to the device. In this case, the username is set to **'admin'** and the password is set to **'cisco'**. You should replace them with the actual username and password of your device.

The **connection** object represents the connection to the specified device and allows you to interact with it using various methods provided by Netmiko.

Note: The connection variable represents an instance of the ConnectHandler class from the Netmiko library. Therefore, the data type of connection would be an object of the ConnectHandler class.

**Step 4: Send an IOS command - send\_command()**

The **send\_command** method in Netmiko is used to send a command to a network device and retrieve the corresponding output. The **send\_command()** is a method associated with the **connection** object created using the **ConnectHandler** class in Netmiko.

You can use the **send\_command()** without **print**, however the **print** option is helpful otherwise you get **\n** instead of line breaks.

We are using the **send\_command()** method on the **connection** object with the parameters given in the previous **ConnectHandler()** method.

**print(connection.send\_command('show ip interface brief'))**

Note: We will be using the **connection** object followed by one of the Netmiko methods throughout these labs.

**Step 5: Disconnect from Netmiko - disconnect()**

The disconnect method in Netmiko is used to close the SSH connection to a network device, ensuring proper cleanup and freeing up system resources.

**connection.disconnect()**

**Step 6: Display the program**

# Import Netmiko library

import netmiko

connection = netmiko.ConnectHandler(ip='192.168.1.1',

device\_type='cisco\_ios',

username='admin',

password='cisco')

print(connection.send\_command('show ip interface brief'))

connection.disconnect()

*Rick's program: netmiko\_1a.py*

Netmiko is very simple to use and really only has a few commands (methods) that we need to use:

1. **ConnectHandler()** - Establishes the SSH connection and is assigned to a variable (e.g. *connection*). The **connection** object represents the connection to the specified device and allows you to interact with it using various methods provided by Netmiko.
   1. *connection***.enable()** - Enter privileged mode if necessary.
   2. *connection.***send\_command()** - Send a Cisco IOS non-configuration command (e.g. show command)
   3. *connection.***send\_config\_set()** - Send a Cisco IOS configuration command (e.g. ipv6 address fe80::1 link-local)
   4. *connection***.save\_config()** – Save the running-config to startup-config
2. *connection***.disconnect()** - Disconnect from the SSH session

Everything else is just Python. Just a little Python is all you need to get started. The more Python you know, the more things you can do.

Our labs do not require much Python knowledge and even if some of the Python is new to you, you can still easily follow and implement the code.

Some of the Python we will use:

* Lists
* Dictionaries
* For loops
* Exception handling (try and except)

We will also use files and JSON. Even if you do not understand the code in detail, you will see how to use it.

**Step 7: Run the program**

Interface IP-Address OK? Method Status Protocol

Embedded-Service-Engine0/0 unassigned YES NVRAM administratively down down

GigabitEthernet0/0 192.168.1.1 YES NVRAM up up

GigabitEthernet0/1 192.168.2.1 YES NVRAM up up

Serial0/0/0 unassigned YES NVRAM administratively down down

Serial0/0/1

**Step 8: Privileged Exec Commands**

Some commands, such as **show running-config** and all configuration commands, require privileged exec level access (**enable secre**t).

We need to do two things:

1. Modify the ConnectHandler() method to include the privileged exec password:

**connection = netmiko.ConnectHandler(ip='192.168.1.1',**

**device\_type='cisco\_ios',**

**username='admin',**

**password='cisco',**

**secret='class'**

**)**

Note: Don’t forget the comma after **password=’cisco’,**

2. Use the **enable()** method on our **connection** object to enter privileged exec mode. This must be prior to using **send\_connection()**.

**connection.enable()**

**Step 9: Use send\_command() to display running-config**

Now modify the **send\_command()** with the parameter, 's**how running-config'**:

**print(connection.send\_command('show running-config'))**

Note: The send\_command() will display the output without using a print function. However, you will see '\n' instead of line breaks. So, using the print function is typically the better option. Try it without the print if you wish:

**connection.send\_command('show running-config')**

**Step 10: Display the program**

# Import Netmiko library

import netmiko

connection = netmiko.ConnectHandler(ip='192.168.1.1',

device\_type='cisco\_ios',

username='admin',

password='cisco',

secret='class'

)

# Enter privileged exec mode

connection.enable()

print(connection.send\_command('show running-config'))

connection.disconnect()

*Rick's program: netmiko\_1b.py*

**Step 11: Run the program**

Building configuration…

Current configuration : 1838 bytes

!

! Last configuration change at 11:45:32 UTC Fri Jun 23 2023

version 15.2

service timestamps debug datetime msec

service timestamps log datetime msec

no service password-encryption

!

hostname R1

!

<output omitted>

!

line con 0

exec-timeout 0 0

logging synchronous

line aux 0

line 2

no activation-character

no exec

transport preferred none

transport output lat pad telnet rlogin lapb-ta mop udptn v120 ssh

stopbits 1

line vty 0 4

login local

transport input ssh

!

scheduler allocate 20000 1000

!

end

Congratulations! Now move on to the next lab…