

EETS 8353 NETWORK AUTOMATION

NETWORK AUTOMATION WITH NETMIKO

PALAK GANDHI

48900571

SPRING 2023



SMU

Agenda

01. Introduction & Features

02. Installation

03. Use Cases

04. Limitations

05. References



Introduction & Features



Netmiko is a powerful open-source library for network automation that simplifies the process of managing network devices such as routers, switches, and firewalls using Python.

- **Vendor-agnostic:** Netmiko supports a wide range of network devices from various vendors, making it a vendor-agnostic tool. This means that you can use the same code to interact with devices from different vendors, simplifying the automation process.
- **SSH and Telnet support:** Netmiko supports both SSH and Telnet protocols for connectivity to network devices. This allows you to use the most appropriate protocol for your environment.



KIRK BYERS

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

Introduction & Features



- **Command execution:** Netmiko allows you to execute commands on network devices and retrieve the output. This can be used for various tasks such as retrieving device information, debugging, and troubleshooting.
- **Configuration management:** Netmiko allows you to manage device configurations by sending commands in configuration mode. This can be used to automate configuration changes, backups, and rollbacks.
- **File transfer:** Netmiko allows you to transfer files to and from network devices. This can be used to automate tasks such as software upgrades and configuration backups.
- **Advanced features:** Netmiko provides advanced features such as enabling and disabling interfaces, configuring VLANs, and sending multiple commands in a single session. This allows you to automate complex network management tasks.

“ Installation & Setup

1. Build the Network and Verify Connectivity through SSH

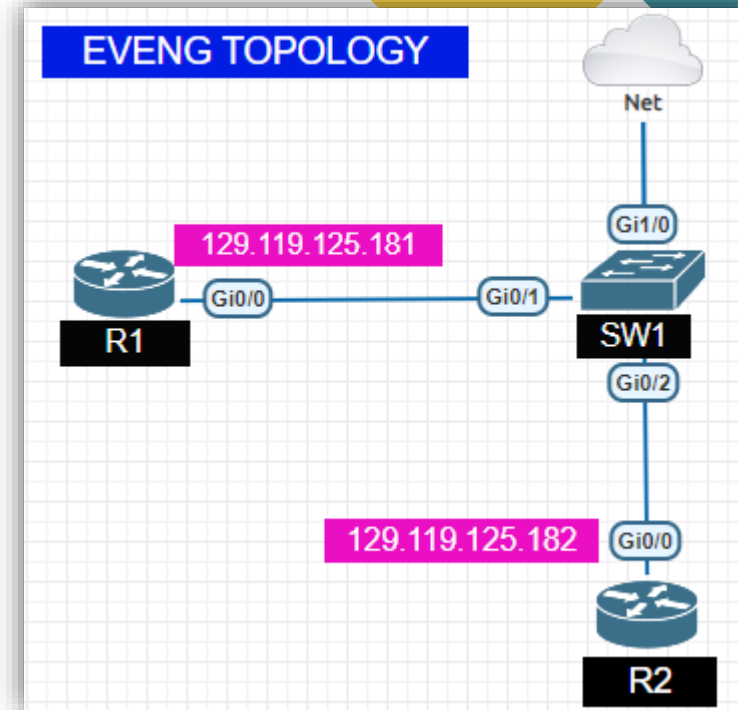
```
PS C:\Users\Navkar> ping 129.119.125.181

Pinging 129.119.125.181 with 32 bytes of data:
Reply from 129.119.125.181: bytes=32 time=26ms TTL=254
Reply from 129.119.125.181: bytes=32 time=21ms TTL=254
Reply from 129.119.125.181: bytes=32 time=30ms TTL=254
Reply from 129.119.125.181: bytes=32 time=34ms TTL=254

Ping statistics for 129.119.125.181:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 21ms, Maximum = 34ms, Average = 27ms
PS C:\Users\Navkar> ping 129.119.125.182

Pinging 129.119.125.182 with 32 bytes of data:
Reply from 129.119.125.182: bytes=32 time=25ms TTL=254
Reply from 129.119.125.182: bytes=32 time=24ms TTL=254
Reply from 129.119.125.182: bytes=32 time=37ms TTL=254
Reply from 129.119.125.182: bytes=32 time=28ms TTL=254

Ping statistics for 129.119.125.182:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 24ms, Maximum = 37ms, Average = 28ms
PS C:\Users\Navkar>
```



“ Installation & Setup

2. Import Netmiko Python Module

- Check Python version
- Check availability of Netmiko
- Install Netmiko using pip
- Verify Netmiko

```
PS C:\Users\Navkar>
PS C:\Users\Navkar> python --version
Python 3.10.7
PS C:\Users\Navkar>
PS C:\Users\Navkar> pip show netmiko
WARNING: Package(s) not found: netmiko
PS C:\Users\Navkar>
PS C:\Users\Navkar> pip install netmiko
Collecting netmiko
  Using cached netmiko-4.1.2-py3-none-any.whl (196 kB)
Requirement already satisfied: textfsm==1.1.2 in c:\use
Requirement already satisfied: scp>=0.13.3 in c:\users\
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\Navkar>
PS C:\Users\Navkar> pip show netmiko
Name: netmiko
Version: 4.1.2
Summary: Multi-vendor library to simplify legacy CLI connections to
Home-page: https://github.com/ktbyers/netmiko
Author: Kirk Byers
Author-email: ktbyers@twb-tech.com
License: MIT
Location: c:\users\navkar\appdata\local\programs\python\python310\l
Requires: ntc-templates, paramiko, pyserial, pyyaml, scp, setuptool
Required-by:
PS C:\Users\Navkar>
```

“ Installation & Setup

3. Create a dictionary representing the device.

By creating this dictionary, the Netmiko library can use these values to establish a connection to the specified network device. The values for these keys will vary depending on the specific device. The supported devices can be found on:

[netmiko/PLATFORMS.md at develop · ktbyers/netmiko · GitHub](#)

```
cisco_device = {  
    'device_type': 'cisco_ios',  
    'host': '129.119.125.181',  
    'username': 'admin',  
    'password': 'cisco',  
    'port': 22,  
}
```

“ Installation & Setup

3. Use Netmiko to Connect to the SSH Service

ConnectHandler is the main entry point into the library. It picks the right class, creates a Netmiko object based on that class, and establishes an SSH connection to the remote device.

We create an instance of the **ConnectHandler** class, passing in the **cisco_device** dictionary using the ****** notation to unpack its contents. The resulting object, **connection**, represents the connection to the device.

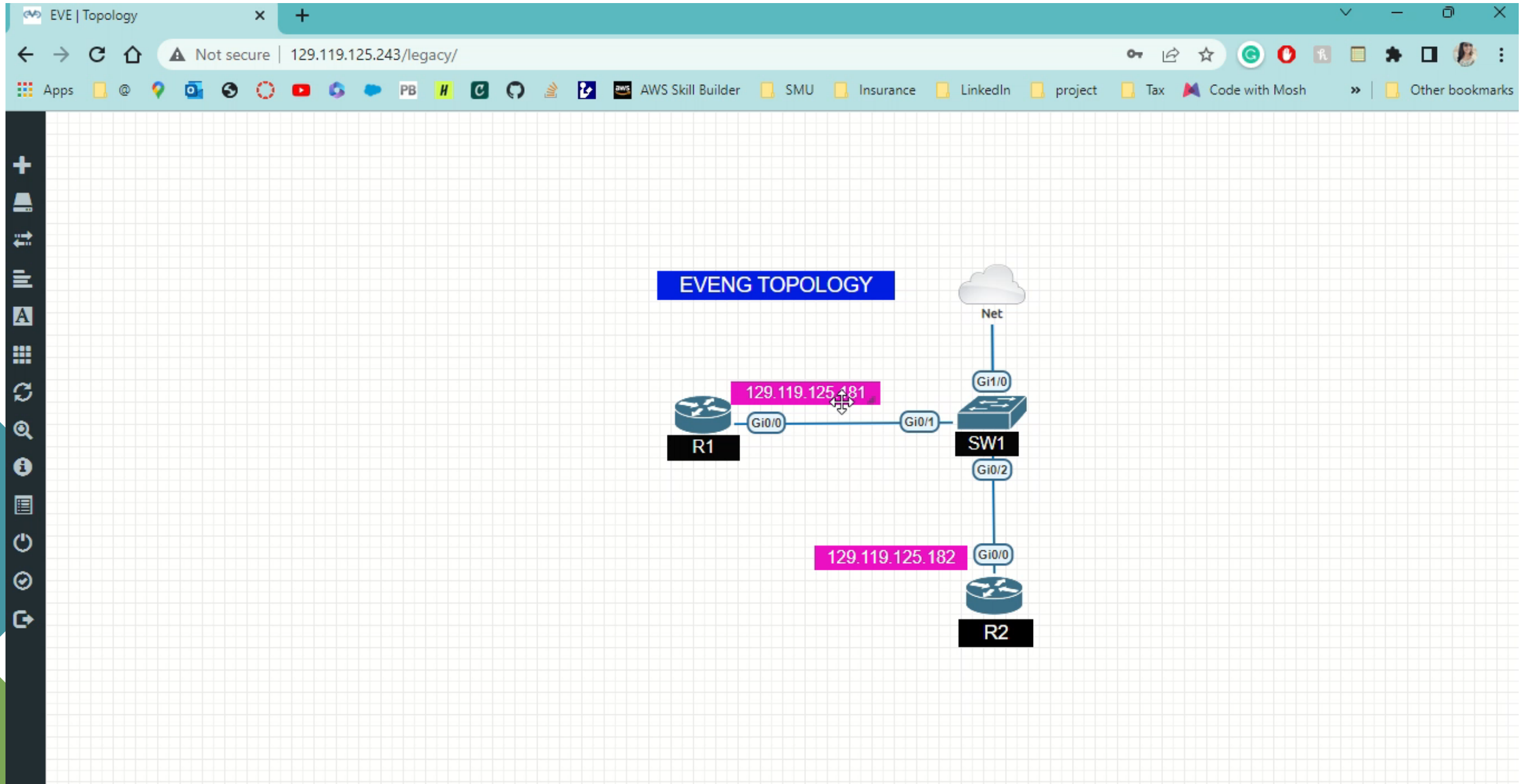
```
# creating an SSH connection to the Cisco device
connection = ConnectHandler(**cisco_device)
```


Use Cases

1. Check installation
2. Connect and verify using single command
3. Run multiple commands on single device
4. Configuration using txt file
5. Run multiple commands for multiple devices
6. How to backup
7. How to handle troubleshoot

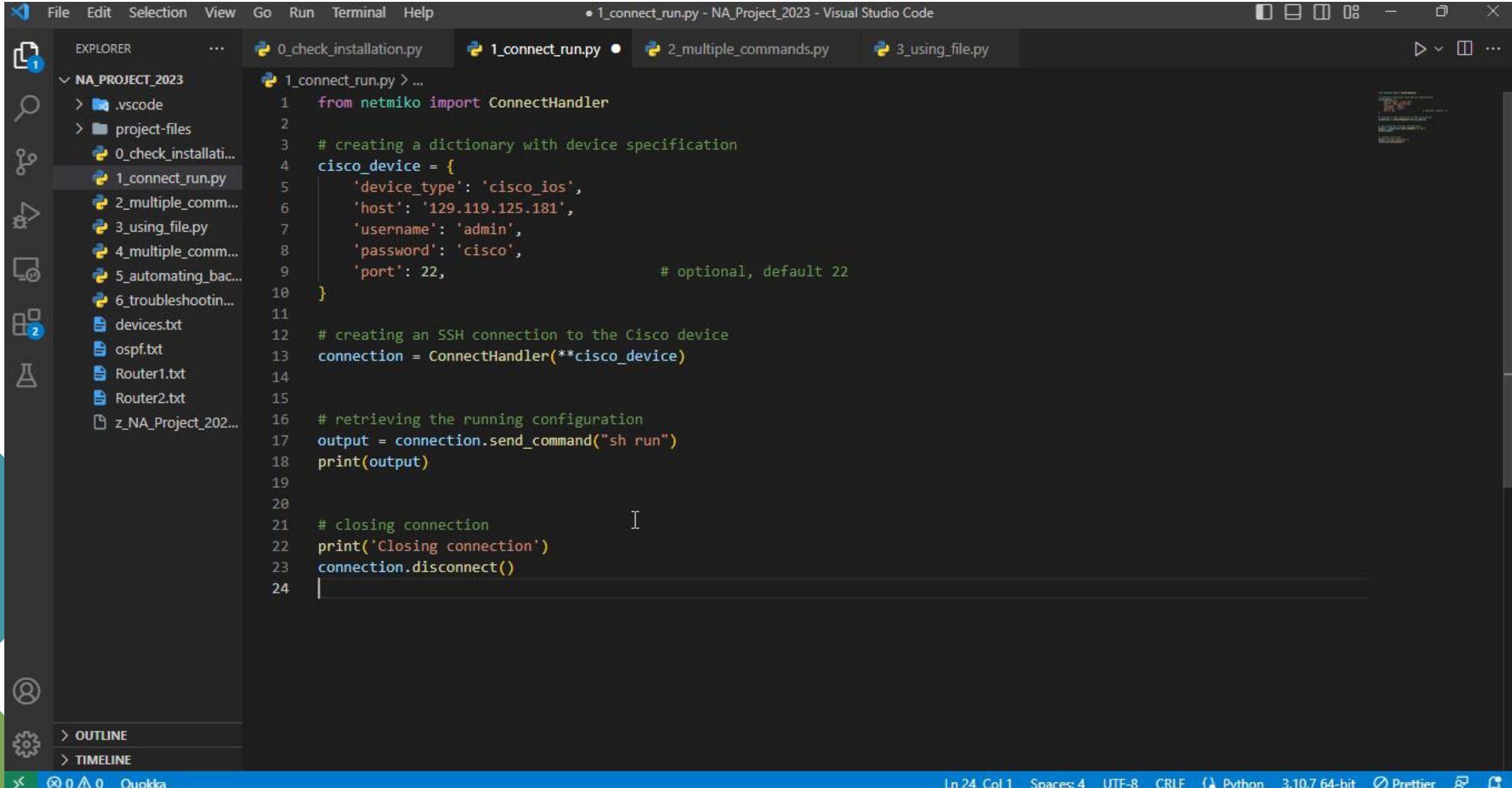
Use Cases

1. Check installation



Use Cases

2. Connect and verify using single command



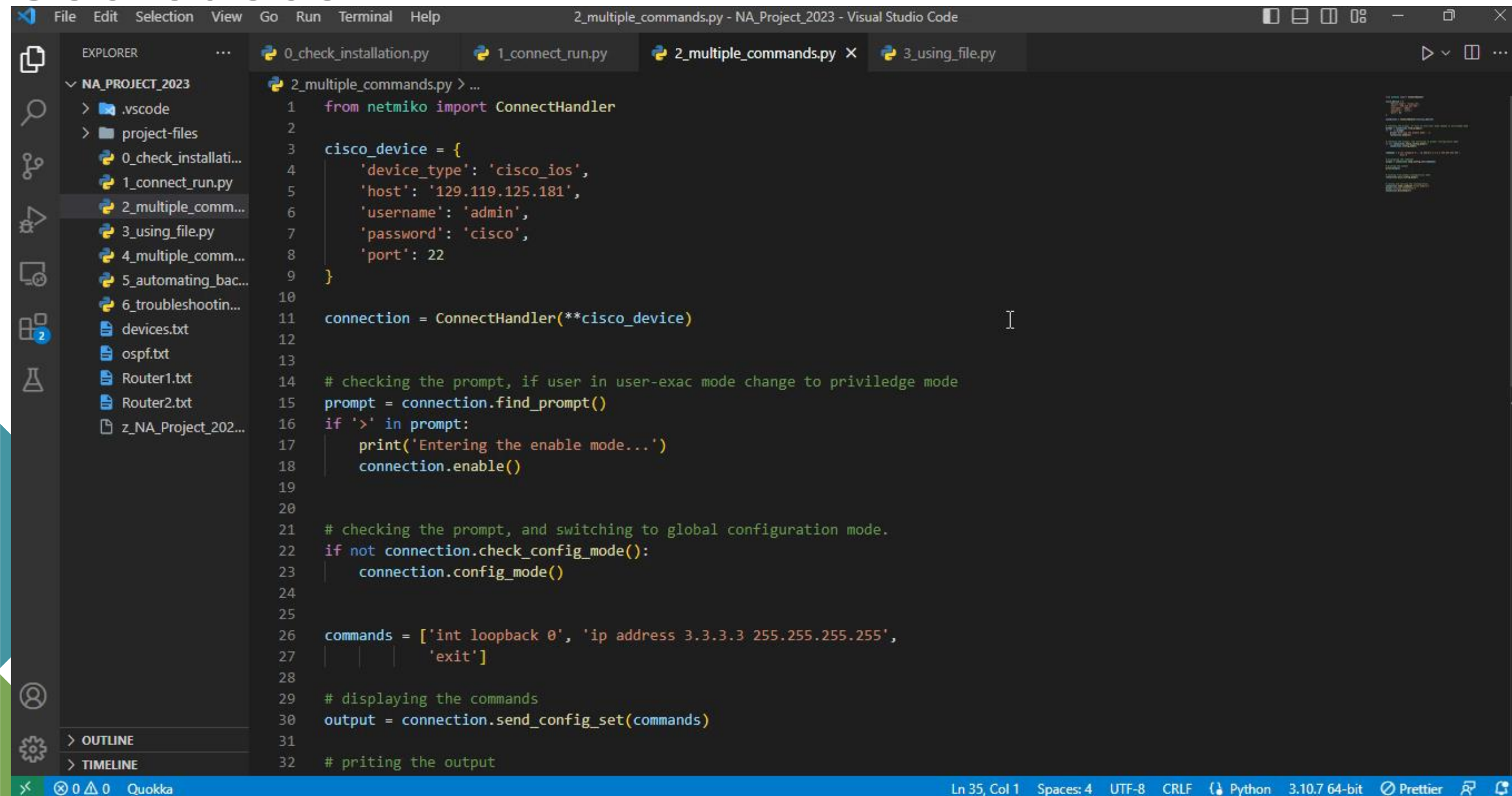
The screenshot displays the Visual Studio Code interface with a project named 'NA_PROJECT_2023'. The Explorer sidebar on the left shows the project structure, including a '.vscode' folder, a 'project-files' folder, and several Python files. The file '1_connect_run.py' is selected and open in the main editor. The code in this file uses the 'netmiko' library to establish an SSH connection to a Cisco device, retrieve the running configuration, and then disconnect. The code is as follows:

```
1 from netmiko import ConnectHandler
2
3 # creating a dictionary with device specification
4 cisco_device = {
5     'device_type': 'cisco_ios',
6     'host': '129.119.125.181',
7     'username': 'admin',
8     'password': 'cisco',
9     'port': 22, # optional, default 22
10 }
11
12 # creating an SSH connection to the Cisco device
13 connection = ConnectHandler(**cisco_device)
14
15
16 # retrieving the running configuration
17 output = connection.send_command("sh run")
18 print(output)
19
20
21 # closing connection
22 print('Closing connection')
23 connection.disconnect()
24
```

The status bar at the bottom indicates the current position is Line 24, Column 1, with 4 spaces, UTF-8 encoding, CRLF line endings, Python 3.10.7 64-bit, and Prettier formatting.

Use Cases

3. Run multiple commands on single device

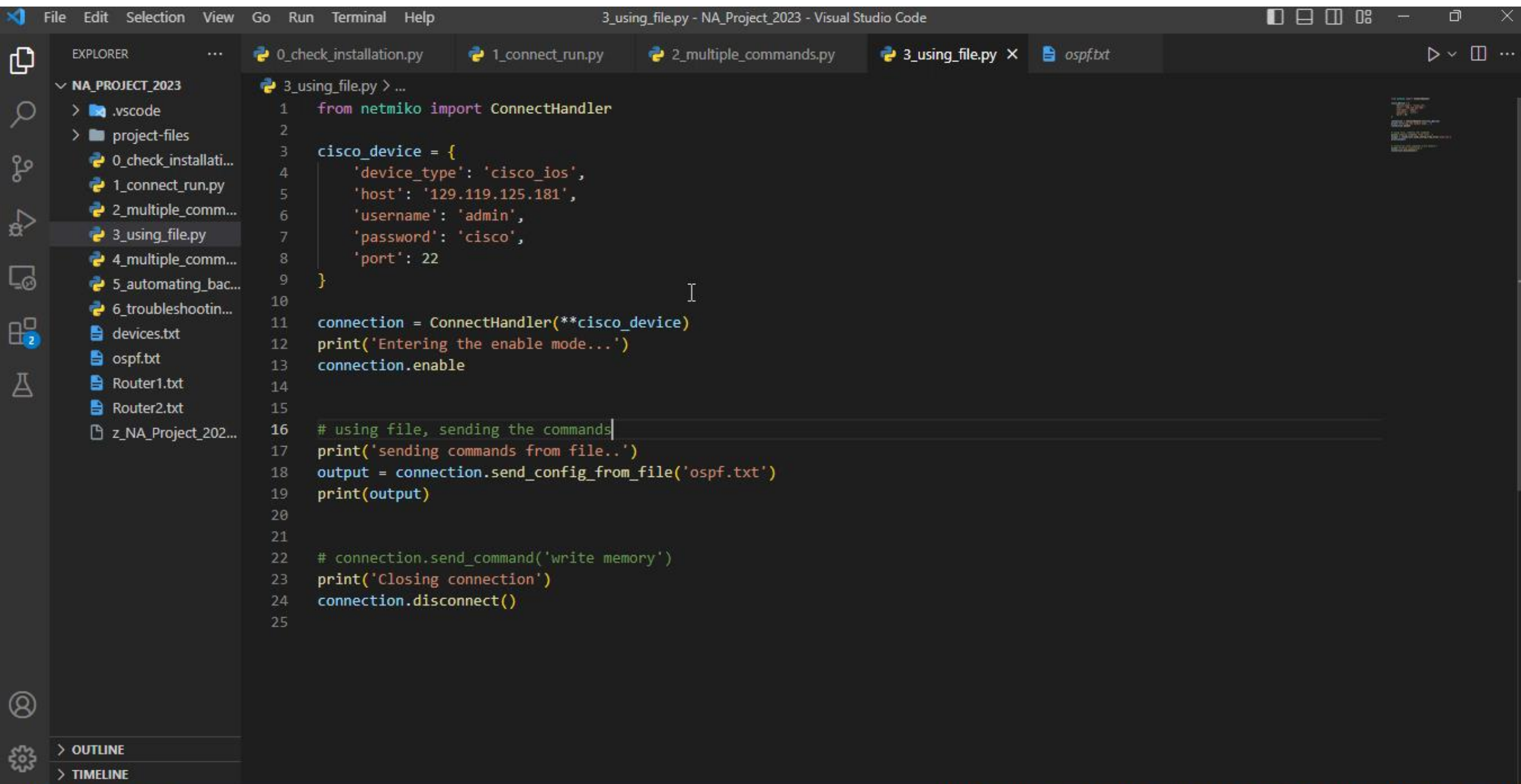


The screenshot shows the Visual Studio Code interface with a project named 'NA_PROJECT_2023'. The Explorer sidebar on the left lists files including '.vscode', 'project-files', and several Python scripts. The main editor window displays the file '2_multiple_commands.py'. The code in this file uses the 'netmiko' library to connect to a Cisco device and execute a list of commands. The script includes comments for each step, such as checking the prompt, enabling privileged mode, switching to global configuration mode, and sending a set of commands. The status bar at the bottom indicates the current cursor position is at line 35, column 1, and the file is encoded in UTF-8 with CRLF line endings.

```
2_multiple_commands.py > ...
1  from netmiko import ConnectHandler
2
3  cisco_device = {
4      'device_type': 'cisco_ios',
5      'host': '129.119.125.181',
6      'username': 'admin',
7      'password': 'cisco',
8      'port': 22
9  }
10
11  connection = ConnectHandler(**cisco_device)
12
13
14  # checking the prompt, if user in user-exac mode change to priviledge mode
15  prompt = connection.find_prompt()
16  if '>' in prompt:
17      print('Entering the enable mode...')
18      connection.enable()
19
20
21  # checking the prompt, and switching to global configuration mode.
22  if not connection.check_config_mode():
23      connection.config_mode()
24
25
26  commands = ['int loopback 0', 'ip address 3.3.3.3 255.255.255.255',
27              'exit']
28
29  # displaying the commands
30  output = connection.send_config_set(commands)
31
32  # priting the output
```

Use Cases

4. Configuration using txt file

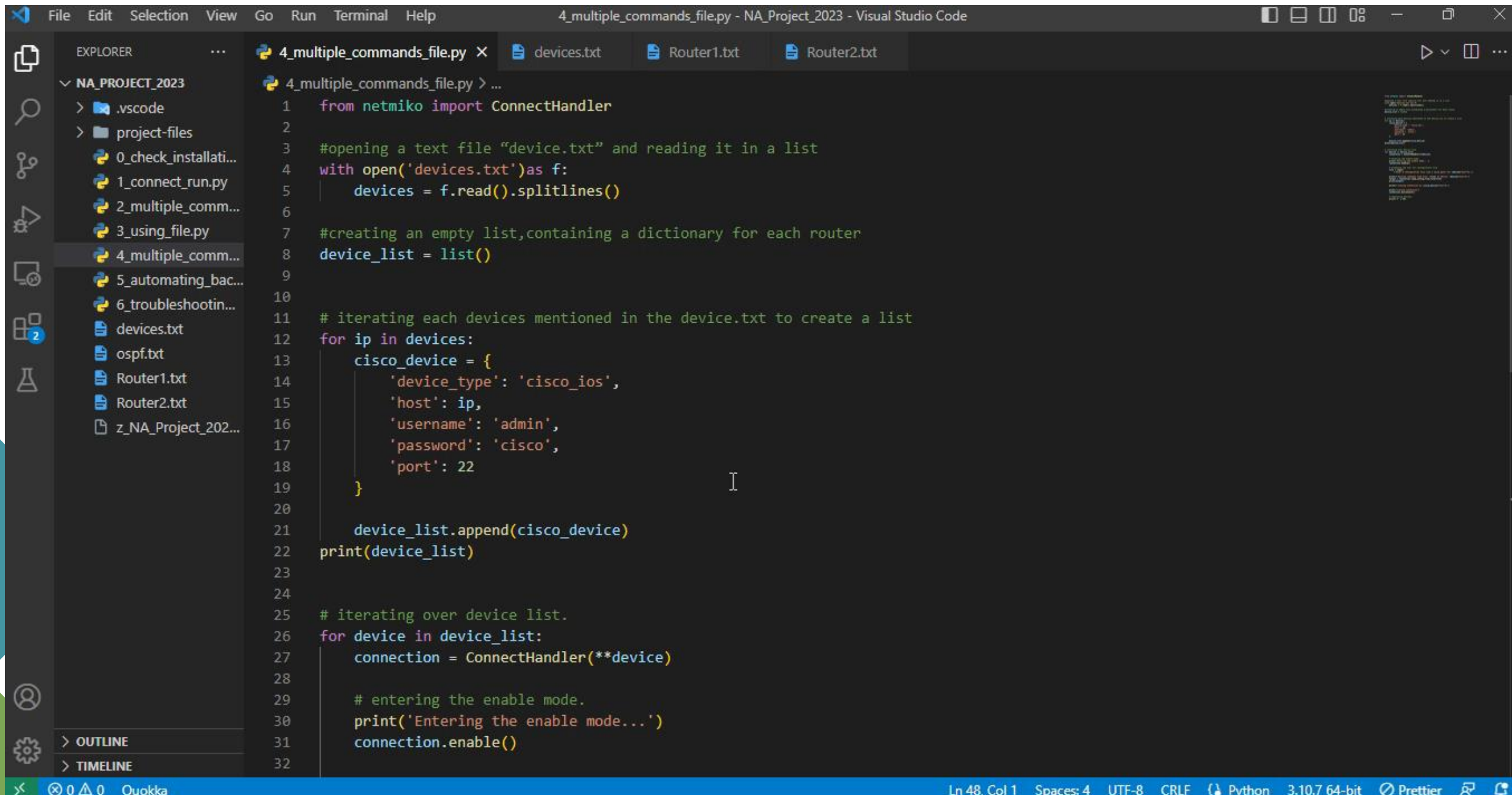


The screenshot displays the Visual Studio Code interface with a project named 'NA_PROJECT_2023'. The Explorer sidebar on the left shows the project structure, including a 'project-files' folder containing several Python scripts and text files. The file '3_using_file.py' is selected and open in the editor. The code in the editor is a Python script that uses the 'netmiko' library to connect to a Cisco device and send configuration commands from a file named 'ospf.txt'. The script includes comments and print statements for debugging.

```
3_using_file.py > ...
1  from netmiko import ConnectHandler
2
3  cisco_device = {
4      'device_type': 'cisco_ios',
5      'host': '129.119.125.181',
6      'username': 'admin',
7      'password': 'cisco',
8      'port': 22
9  }
10
11  connection = ConnectHandler(**cisco_device)
12  print('Entering the enable mode...')
13  connection.enable
14
15
16  # using file, sending the commands
17  print('sending commands from file..')
18  output = connection.send_config_from_file('ospf.txt')
19  print(output)
20
21
22  # connection.send_command('write memory')
23  print('Closing connection')
24  connection.disconnect()
25
```


Use Cases

5. Run commands for multiple devices



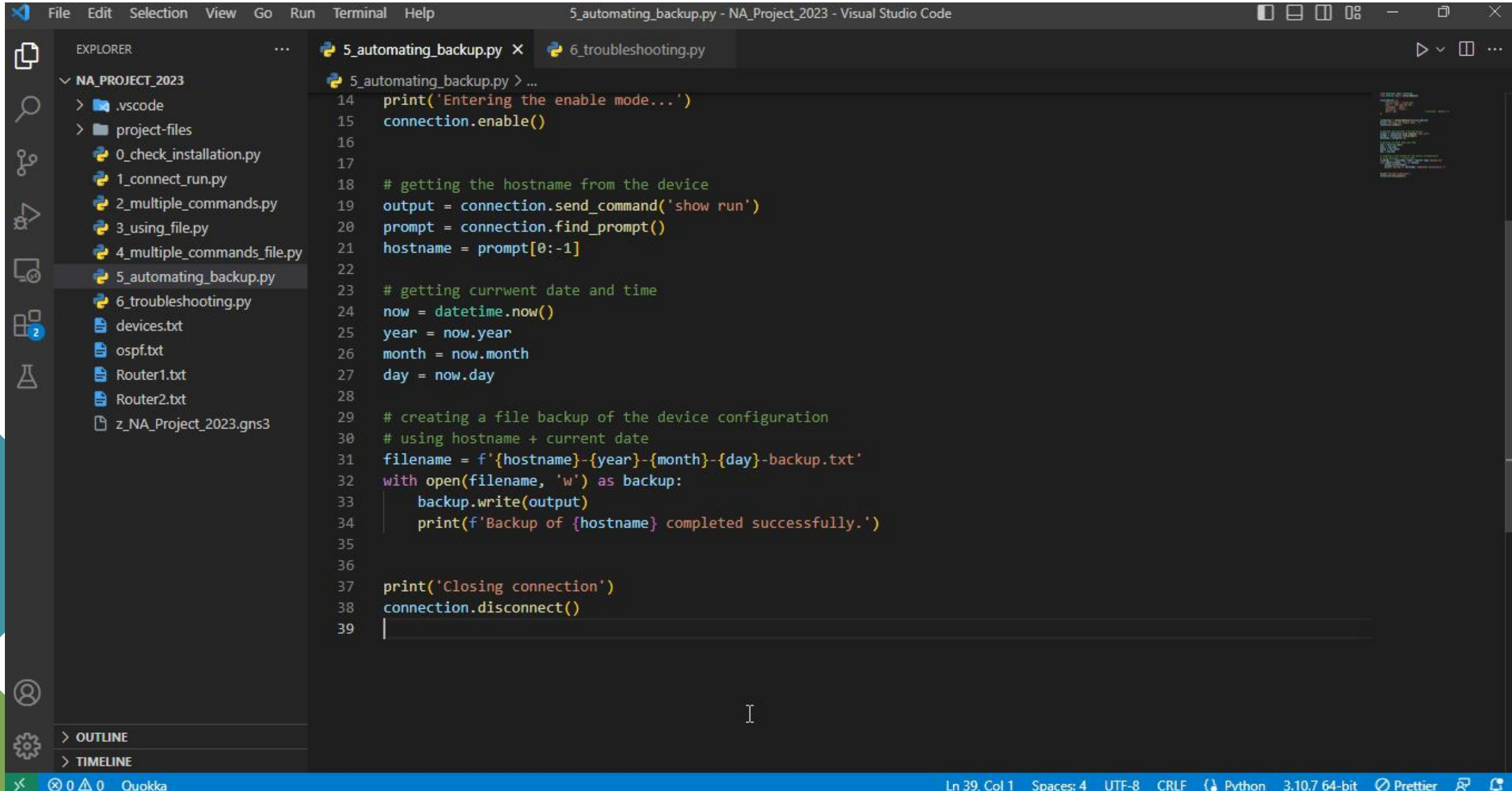
The screenshot displays the Visual Studio Code interface with a Python script named `4_multiple_commands_file.py` open. The Explorer sidebar on the left shows a project named `NA_PROJECT_2023` containing several files, including `devices.txt`, `Router1.txt`, and `Router2.txt`. The main editor area shows the following Python code:

```
1 from netmiko import ConnectHandler
2
3 #opening a text file "device.txt" and reading it in a list
4 with open('devices.txt') as f:
5     devices = f.read().splitlines()
6
7 #creating an empty list,containing a dictionary for each router
8 device_list = list()
9
10
11 # iterating each devices mentioned in the device.txt to create a list
12 for ip in devices:
13     cisco_device = {
14         'device_type': 'cisco_ios',
15         'host': ip,
16         'username': 'admin',
17         'password': 'cisco',
18         'port': 22
19     }
20
21     device_list.append(cisco_device)
22 print(device_list)
23
24
25 # iterating over device list.
26 for device in device_list:
27     connection = ConnectHandler(**device)
28
29     # entering the enable mode.
30     print('Entering the enable mode...')
31     connection.enable()
32
```

The status bar at the bottom indicates the current position is Line 48, Column 1, with 4 spaces, UTF-8 encoding, CRLF line endings, Python 3.10.7 64-bit, and the Prettier formatter.

Use Cases

6./7. How to backup & troubleshoot



The screenshot shows the Visual Studio Code interface with a project named 'NA_Project_2023'. The Explorer panel on the left shows the project structure, including a 'project-files' folder and several Python scripts. The main editor displays the code for '5_automating_backup.py', which is a Python script designed to automate the backup of a network device configuration. The script includes comments in green and code in various colors. It starts by entering enable mode, then sends a 'show run' command to retrieve the configuration. It then extracts the hostname from the prompt and gets the current date and time. Finally, it creates a backup file named 'hostname-year-month-day-backup.txt' and writes the configuration to it. The script concludes by closing the connection and printing a success message.

```
File Edit Selection View Go Run Terminal Help 5_automating_backup.py - NA_Project_2023 - Visual Studio Code

EXPLORER
NA_PROJECT_2023
├── .vscode
├── project-files
│   ├── 0_check_installation.py
│   ├── 1_connect_run.py
│   ├── 2_multiple_commands.py
│   ├── 3_using_file.py
│   ├── 4_multiple_commands_file.py
│   ├── 5_automating_backup.py
│   ├── 6_troubleshooting.py
│   ├── devices.txt
│   ├── ospf.txt
│   ├── Router1.txt
│   ├── Router2.txt
│   └── z_NA_Project_2023.gns3
└── OUTLINE
    └── TIMELINE

5_automating_backup.py > ...
14 print('Entering the enable mode...')
15 connection.enable()
16
17
18 # getting the hostname from the device
19 output = connection.send_command('show run')
20 prompt = connection.find_prompt()
21 hostname = prompt[0:-1]
22
23 # getting currwent date and time
24 now = datetime.now()
25 year = now.year
26 month = now.month
27 day = now.day
28
29 # creating a file backup of the device configuration
30 # using hostname + current date
31 filename = f'{hostname}-{year}-{month}-{day}-backup.txt'
32 with open(filename, 'w') as backup:
33     backup.write(output)
34     print(f'Backup of {hostname} completed successfully.')
35
36
37 print('Closing connection')
38 connection.disconnect()
39 |
```

Ln 39, Col 1 Spaces: 4 UTF-8 CRLF Python 3.10.7 64-bit Prettier

Limitations

Unsupported devices: Netmiko supports a wide range of network devices from various vendors, but there may be some devices that are not yet supported. If the device is not supported by Netmiko, we may need to use another library or write custom code to automate its configuration.

Performance limitations: Netmiko relies on SSH or Telnet connections to communicate with network devices, which can introduce some performance overhead. If we need to perform high-volume or real-time data processing, we may need to consider other approaches, such as using SNMP or a dedicated network management system.

Unmanageable for large networks: Netmiko may not be suitable for extremely large or complex networks, as it may become cumbersome to manage and scale.

Security concerns: While Netmiko provides secure SSH connections to network devices, there may be security risks associated with using Python scripts for network automation. In some cases, we may need to use more secure and auditable approaches, such as using configuration management tools or running automation scripts in a dedicated environment.

Limited functionality: While Netmiko provides a rich set of features for interacting with network devices, it may not cover every possible use case or scenario. In some cases, we may need to use other libraries or tools to complement Netmiko's functionality.



“References

- <https://github.com/ktbyers/netmiko>
- https://pyneng.readthedocs.io/en/latest/book/18_ssh_telnet/netmiko.html
- <https://ktbyers.github.io/netmiko/docs/netmiko/index.html>
- https://www.youtube.com/watch?v=76MFWqhwFfs&ab_channel=NetworktoCode
- <https://pypi.org/project/netmiko/>



Thank you

