**Знакомство с Paramiko и Netmiko**

* [**telnetlib**](https://docs.python.org/3/library/telnetlib.html) - встроенная библиотека, она используется для обеспечения доступа к сетевому устройству и отправке команд по Telnet протоколу (не является безопасным).
* [**pexpect**](https://pexpect.readthedocs.io/en/stable/) - внешняя библиотека работает для передачи данных по протоколам: ssh, telnet, sftp и др. Считается универсальным инструментом для передачи команд как в ОС, так и на сетевое оборудование. Логика работы основана на ожидании получения соответствующей строки от устройства и дальнейшей отправки команды.
* [**paramiko**](http://www.paramiko.org/) - внешняя библиотека, обеспечивающая организацию SSH-соединения. Работает в режиме клиента и сервера, соответственно в качестве передачи данных может использоваться только протокол SSH.
* [**netmiko**](https://github.com/ktbyers/netmiko) - внешняя библиотека от сообщества сетевых инженеров и администраторов, была разработана специально для взаимодействия с сетевым оборудованием различных производителей (адаптация под сетевую ОС). Netmiko включает в себя paramiko, описанный раннее.

Автоматизация работы сетевого администратора. Часть 3. Знакомство с Paramiko

**Автор:** [Никита Турков](https://nag.ru/search?query=%D0%B0%D0%B2%D1%82%D0%BE%D1%80:%D0%9D%D0%B8%D0%BA%D0%B8%D1%82%D0%B0%20%D0%A2%D1%83%D1%80%D0%BA%D0%BE%D0%B2)

Предыдущие статьи цикла:

[Автоматизация работы сетевого администратора](https://nag.ru/articles/article/108040/avtomatizatsiya-rabotyi-setevogo-administratora.html).  
[Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib](https://nag.ru/articles/article/108074/avtomatizatsiya-rabotyi-setevogo-administratora-chast-2-prakticheskie-keysyi-s-telnetlib.html).

Всех приветствую, продолжаем рассматривать полезные инструменты для работы сетевого инженера или администратора. Для тех, кто только начал изучение темы с текущей статьи, напомню что мы поговорили о концепции SDN и достаточно подробно рассмотрели инструмент TelnetLib с практическими примерами. Буквально после прочтения 2х статей и некоторого навыка в IT, вы уже можете автоматизировать рутинные процессы по настройке вашего сетевого оборудования.

В рамках взаимодействия с сетевыми устройствами, есть различные нюансы, которые плохо реализованы в предустановленных модулях языка Python 3, однако существуют библиотеки от сообщества и энтузиастов, которые подойдут под ваши задачи. Сегодня мы рассмотрим модуль Paramiko.

SSH и Paramiko

[Paramiko](http://docs.paramiko.org/en/stable/index.html) - это реализация протокола SSHv2 на Python. Paramiko предоставляет функциональность клиента и сервера. В рамках статьи будет рассматриваться лишь функциональность клиента, серверную часть возможно изучить самостоятельно.

Библиотека Paramiko не входит в список стандартных модулей для Python 3, поэтому ее требуется установить:

**pip install paramiko**

В Python3 есть отличная возможность тестировать ваши скрипты, отправляя одиночные команды, для этого достаточно лишь вызвать сам интерпретатор:

**python3**

**Логика работы Paramiko:**

1. создание клиента подключения SSH;
2. настройка клиента подключения SSH;
3. выполнение подключения SSH;
4. переход в интерактивный режим отправки команд для подключения SSH.

**Создание клиента SSH:**

**In [1]: import paramiko**  
**In [2]: client = paramiko.SSHClient()**  
**In [3]: client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())**  
**In [4]: client.connect(hostname="10.10.10.1", username="admin", password="admin",look\_for\_keys=False, allow\_agent=False)**  
**In [5]: ssh = client.invoke\_shell()**

**Комментарии к каждой из строк:**

In [1]: импорт библиотеки**Paramiko**

In [2]: присваивание переменной client класса**SSHClient** cо всеми атрибутами.

In [3]: необязательная строка кода служит для указания какую политику использовать клиенту при подключение к неизвестному серверу в рамках SSH. В примере **paramiko.AutoAddPolicy()** автоматически добавляет новое имя хоста и ключ в локальный объект HostKeys.

In [4]: Метод connect выполняет подключение к SSH-серверу и аутентифицирует подключение. Его параметры: **look\_for\_keys** - режим аутентификации по ключам (True/False); **allow\_agent -**режим подключения через локальный SSH-агент OC (True/False).

**Метод send:**

Отправка указанной строки и возврат количества переданных байт, если значение "0" - сессия закрыта и не удалось отправить команду.

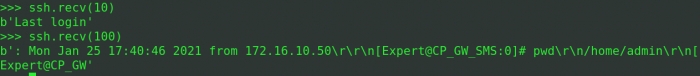
**In [6]: ssh.send("pwd\n")  
"> 4**

В нашем примере мы отправили команду "pwd" c целью узнать текущую директорию и получили ответ в виде 4 переданных байт.

**Метод recv:**

Получение данных с терминала устройства по построенному каналу SSH. Данные возвращаются в виде строки, вы указывается количество запрашиваемых данных в виде числа байт. Если возвращается число 0, канал SSH будет автоматически закрыт.

**In [6]: ssh.recv(100)**

[](https://nag.ru/upload/images/20210128-0001.png)

**Метод close:**

Закрытие SSH - сессии

**In [7]: ssh.close()**

**Скрипт с выводом конфигурации на экран**

Для закрепления рассмотрим скрипт по выводу "sh run" в Paramiko. После # или двойных кавычек будут следовать комментарии.

"Импорт различных библиотек для скрипта"

**import paramiko**  
**import time**  
**import socket**  
**from pprint import pprint**  
**import re**

"Функция по отправке команд "

**def send\_show\_command(ip,**  
**username,**  
**password,**  
**enable,**  
**command,**  
**max\_bytes=60000,**  
**short\_pause=1,**  
**long\_pause=5,):**

**cl = paramiko.SSHClient()**# создание класса SSH-клиента  
**cl.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())**

"Передача аргументов в метод connect"

**hostname=ip,**  
**username=username,**  
**password=password,**  
**look\_for\_keys=False,**  
**allow\_agent=False,**  
**)**

"Cоздание интерактивного подключения и вход в режим enable"

**ssh.send("enable\n")**  
**ssh.send(enable + "\n")**  
**time.sleep(short\_pause)**  
**ssh.recv(max\_bytes)**

**result = {}**

"Отправка команд в цикле"

**ssh.send(f"{command}\n")**  
**ssh.settimeout(5)**

**output = ""**

"Вывод за счет цикла всей информации из терминала"

**try:**  
**page = ssh.recv(max\_bytes).decode("utf-8")**  
**output += page**  
**time.sleep(0.5)**  
**except socket.timeout:**  
**break**

"Если требуется перейти в режим More"

**ssh.send(" ")**  
**output = re.sub(" +--More--| +\x08+ +\x08+", "\n", output)**  
**result[command] = output**

**return result**# возврат результата вывода

"Выполняется, если запускается непосредственно скрипт".

**if \_\_name\_\_ == "\_\_main\_\_":**  
**commands = ["sh run"]**  
**result = send\_show\_command("192.168.100.1", "admin", "password", "cisco", commands)**  
**pprint(result, width=120)**

Конечно, для новичков в Python будет достаточно сложно разобраться со всеми строками кода, здесь используются различные возможности ЯП. По факту вам достаточно лишь изменить конструкцию после **if \_\_name\_\_ == "\_\_main\_\_":**

Где вы будете вызывать фунцию **send\_show\_command("192.168.100.1", "admin", "password", commands)**

Произойдет соответственно передача аргументов: IP-адрес, логин, пароль, команды.

Автоматизация 4. Netmiko

**Автор:** [Никита Турков](https://nag.ru/search?query=%D0%B0%D0%B2%D1%82%D0%BE%D1%80:%D0%9D%D0%B8%D0%BA%D0%B8%D1%82%D0%B0%20%D0%A2%D1%83%D1%80%D0%BA%D0%BE%D0%B2)

Предыдущие статьи цикла

* [Автоматизация работы сетевого администратора.](https://nag.ru/articles/article/108040/avtomatizatsiya-rabotyi-setevogo-administratora.html)
* [Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib.](https://nag.ru/articles/article/108074/avtomatizatsiya-rabotyi-setevogo-administratora-chast-2-prakticheskie-keysyi-s-telnetlib.html)
* [Автоматизация работы сетевого администратора. Часть 3. Знакомство с Paramiko](https://nag.ru/articles/article/108094/avtomatizatsiya-rabotyi-setevogo-administratora-chast-3-znakomstvo-s-paramiko.html).

Приветствую читателей цикла об автоматизации работы с сетевым оборудованием, после чуть затянувшегося перерыва мы продолжаем писать о различных программных инструментах . В конце предыдущей статьи, в которой мы познакомились с Paramiko, был упомянут модуль, разработанный сетевыми инженерами для сетевых инженеров.

Введение

Итак, [Netmiko](https://github.com/ktbyers/netmiko" \t "_blank) - это Python библиотека для работы с сетевыми устройствами, в качестве инициализации соединений она использует Paramiko, но содержит много встроенных методов и классов "под капотом".

Библиотека не входит в список стандартных модулей для Python 3, поэтому ее требуется установить:

**pip install netmiko**

В первую очередь рекомендовано ознакомиться со [списком](https://ktbyers.github.io/netmiko/PLATFORMS.html)поддерживаемых сетевых ОС, из стабильных это:

* Arista vEOS
* Cisco ASA
* Cisco IOS
* Cisco IOS-XE
* Cisco IOS-XR
* Cisco NX-OS
* Cisco SG300
* HP ProCurve
* Juniper Junos
* Linux

**Кратко об истории создания:**

Разработчики, в частности сетевые инженеры, часто сталкивались с архитектурными (системными) ограничениями того или иного сетевого оборудования при написание различных скриптов, что не позволяло унифицировать его применение и вызывало ошибки выполнения.

Например, HP ProCurve свитчи выводят символы в ANSI-кодировке или Cisco требует дополнительные режимы в терминале (enable, configure terminal ) и пр. Это побудило написать библиотеку Netmiko, где уже будет задана логика взаимодействия с сетевым устройством в зависимости от его ОС.

Синтаксис и логика

Напомню, в Python3 есть удобная возможность тестировать команды, для этого достаточно лишь вызвать сам интерпретатор:

**python3**

Первично, нужно импортировать функцию ConnectHandler и подготовить переменную типа словарь, в котором будут заданы необходимые параметры для SSH-подключения.

**In [1]: from netmiko import ConnectHandler**

**In [2]: cisco = {**  
**...:   "device\_type": "cisco\_ios",**  
**...:   "host": "cisco.domain.com",**  
**...:   "username": "admin",**  
**...:   "password": "cisco123",**  
**...:   }**

После того, как определили "device\_type": "cisco\_ios", можем создавать соединение с устройством. Список поддерживаемых видов ОС доступен по [ссылке.](https://github.com/ktbyers/netmiko/blob/master/netmiko/ssh_dispatcher.py#L70)

Для удобства передадим функцию в переменную "net\_connect", ее будет использовать далее при отправке команд и т.д.

**In [3]: net\_connect = ConnectHandler(\*\*cisco)**

С помощью \*\* мы передаем ключ-значение в функцию ConnectHandler, более подробно о теории по [ссылке](https://book.pythontips.com/en/latest/args_and_kwargs.html). Альтернативно возможно отправить параметры функции вручную:

**net\_connect2 = ConnectHandler(device\_type="cisco\_ios", host="cisco.domain.com", username="admin", password="cisco123")**

После создания переменной  "net\_connect" и загрузки в нее параметров, можно убедиться что SSH-соединение установлено:

**In [5]: net\_connect.find\_prompt()**  
**Out[5]: "cisco3#"**

Получен ответ (Out[5]:) от сетевого устройства, сессия активна, находимся в режиме enable (Cisco OS).

Сбор данных с одного устройства

В качестве первой задачи после подключения, рассмотрим получение вывода результата выполнения команды на экране пользователя (инициатора скрипта). Благодаря прописанной логике, в Netmiko достаточно вызвать следующий метод:

**In [6]: output = net\_connect.send\_command("show ip int brief")**  
**In [7]: print(output)**

[Изображение выглядит как текст

Автоматически созданное описание](https://nag.ru/upload/images/20210309-0001.png)

Что произошло внутри ? В строке 6 была определена переменная "оutput", которой был передан класс "net\_connect" - с данными о подключение и вызван метод "send\_command".

Со стороны сетевого устройства: произошла успешная инициализация SSH-соединения, вход в режим enable и отправка команды "show ip int brief" в терминал.Далее осталось лишь записать вывод в переменную "оutput"и вывести ее на экран.

Отработает ли метод, если внутри команды задать еще одно условие для выполнения?

**In [8]: output = net\_connect.send\_command("show run | inc logging")**  
**In [9]: print(output)**

[](https://nag.ru/upload/images/20210309-0003.png)

В In [8] передана команда с просмотром текущего конфига (show run) и ее вывод передан в конвейер Pipeline для команды "inc logging". Соответственно, результат представлен на скриншоте, запрашиваемый параметр выведен.

Сбор данных с нескольких устройств

После того, как рассмотрели сбор данных с одного устройства, выполним задачу по опросу нескольких сетевых устройств, причем в топологии будут различные вендоры (сетевые ОС).

In [1]: from netmiko import ConnectHandler #Импорт библиотеки  
"Создание словарей с информацией об устройствах"

In [2]: cisco3 = {  
   ...:     "device\_type": "cisco\_ios",  
   ...:     "host":   "cisco3.domain.com",  
   ...:     "username": "admin",  
   ...:     "password": "cisco123",  
   ...: }  
   ...:  
In [3]: cisco\_asa = {  
   ...:     "device\_type": "cisco\_asa",  
   ...:     "host": "10.10.10.88",  
   ...:     "username": "admin",  
   ...:     "password": "cisco123",  
   ...:     "secret": "cisco123",  
   ...: }  
   ...:  
In [4]: cisco\_xrv = {  
   ...:     "device\_type": "cisco\_xr",   
   ...:     "host": "10.10.10.77",   
   ...:     "username": "admin",   
   ...:     "password": "cisco123",   
   ...: }

Далее нам понадобится уже известный цикл FOR, с помощью которого мы сможем последовательно подключаться к каждому из сетевых устройств и предоставлять вывод:

for a\_device in all\_devices:   
    ...:     net\_connect = ConnectHandler(\*\*a\_device)   
    ...:     output = net\_connect.send\_command("show arp")   
    ...:     print(f"\n\n--------- Device {a\_device["device\_type"]} ---------")   
    ...:     print(output)   
    ...:     print("--------- End ---------")

В примере будет запрошена таблица ARP:

[Изображение выглядит как текст

Автоматически созданное описание](https://nag.ru/upload/images/20210309-0005.png)

Задание настроек конфигурации

В завершение сегодняшней статьи рассмотрим еще одну задачи по формированию списка VLAN на 3 сетевых коммутаторах:

**"Формирование словарей с данными о сетевых устройствах"**

iosv\_l2\_s1 = {  
    "device\_type": "cisco\_ios",  
    "ip": "192.168.122.72",  
    "username": "david",  
    "password": "cisco"  
}  
iosv\_l2\_s2 = {  
    "device\_type": "cisco\_ios",  
    "ip": "192.168.122.82",  
    "username": "david",  
    "password": "cisco"  
}  
iosv\_l2\_s3 = {  
    "device\_type": "cisco\_ios",  
    "ip": "192.168.122.83",  
    "username": "david",  
    "password": "cisco"  
}  
all\_devices = [iosv\_l2\_s1, iosv\_l2\_s2, iosv\_l2\_s3]**# Запись словарей в список**

**"Запуск цикла для последовательного подключения к оборудованию"**  
for devices in all\_devices:  
    net\_connect = ConnectHandler(\*\*devices) **# считывание пары ключ-значение**  
    for n in range (2,21): **# внутренний цикл от 2 до 21**  
       print ("Creating VLAN " + str(n)) **# вывод на экран строки о создание VLAN с номером**  
       config\_commands = ["vlan " + str(n), "name Python\_VLAN " + str(n)] **# формирование команды для отправки**  
       output = net\_connect.send\_config\_set(config\_commands)**# отправка команды на устройство**  
       print (output) **# вывод на экран переданной конфигурации**

Небольшая шпаргалка по методам взаимодействия с оборудованием:

* net\_connect.send\_command() - отправка единичной команды, по умолчанию возвращает вывод;
* net\_connect.send\_command\_timing() - отправка команды по таймингу, по умолчанию возвращает вывод;
* net\_connect.send\_config\_set() - отправка списка команд;
* net\_connect.send\_config\_from\_file() - отправка конфигурации из указанного файла;
* net\_connect.save\_config() - сохранить текущую конфигурацию в режиме  startup-config;
* net\_connect.enable() - Вход в enable mode;
* net\_connect.disconnect() - Закрытие соединения.

Aвтоматизация 5. Netmiko на практике

**Автор:** [Никита Турков](https://nag.ru/search?query=%D0%B0%D0%B2%D1%82%D0%BE%D1%80:%D0%9D%D0%B8%D0%BA%D0%B8%D1%82%D0%B0%20%D0%A2%D1%83%D1%80%D0%BA%D0%BE%D0%B2)

Предыдущие статьи цикла:

* [Автоматизация работы сетевого администратора](https://nag.ru/articles/article/108040/avtomatizatsiya-rabotyi-setevogo-administratora.html)
* [Автоматизация работы сетевого администратора, часть 2. Практические кейсы с Telnetlib](https://nag.ru/articles/article/108074/avtomatizatsiya-rabotyi-setevogo-administratora-chast-2-prakticheskie-keysyi-s-telnetlib.html)
* [Автоматизация работы сетевого администратора. Часть 3. Знакомство с Paramiko](https://nag.ru/articles/article/108094/avtomatizatsiya-rabotyi-setevogo-administratora-chast-3-znakomstvo-s-paramiko.html)
* [Автоматизация 4. Netmiko](https://nag.ru/articles/article/108267/avtomatizatsiya-4-netmiko.html)

Всех приветствую, продолжаем осваивать программные инструменты и автоматизировать рутинные процессы сетевого администрирования. Сегодняшняя статья будет в основном про практику, будут следовать примеры и описания работы скриптов, все также изучаем Netmiko и Python (необходимый минимум).

Обработка ошибок

До сих пор мы рассматривали различные решения и не касались работы с ошибками при работе программы со стороны пользователя или сервера (в нашем случае - сетевое оборудование).

Зачастую логика скрипта может быть написана правильно, но он все равно не отработает, предоставив специфический вывод об ошибке. В случае конфигурирования более 1-го сетевого узла, это может привести к выходу из программы и потери конечного ожидаемого результата.

Возможные причины сбоя вашего скрипта:

1. Некорректные аутентификационные данные.
2. Истек таймаут для подключения.
3. Прерывание передачи трафика.
4. Недоступность порта SSH со стороны сервера.
5. Различные другие ошибки.

Согласитесь, было бы удобно, каким-либо образом обрабатывать такие ошибки и сообщать пользователю о конкретной проблеме, а что даже наиболее важно - не прерывать выполнение программы далее.

Для этого в Python есть конструкция try / except.

try:  
  BLOCK  
except CODE\_ERROR as err:  
    print("OS error: {0}".format(err))

То есть, в блоке "Try" мы пытаемся выполнить какой-либо код , он может вызвать ошибку, тогда мы пробуем ее обработать и получить о ней вывод - с помощью блока "Except". При этом скрипт продолжит свою работу далее, если это не последние шаги в программе.

Дополнительно есть операторы:

"Break" - принудительный выход из цикла, соответственно далее выполняется сама программа.

"Continue" - возвращает управление в начало цикла, следовательно позволяет пропустить оставшиеся строки в цикле и перейти к следующей итерации в нем же.

Более подробно о подходе по обработке ошибок в официальном мануале - [ССЫЛКА](https://docs.python.org/3/tutorial/errors.html).

Вернемся к нашей библиотеке Netmiko и рассмотрим скрипт, который будет проверять доступность сетевого устройства:

#!/usr/bin/env python  
"Импорт необходимых библиотек"  
from getpass import getpass  
from netmiko import ConnectHandler  
from netmiko.ssh\_exception import NetMikoTimeoutException  
from paramiko.ssh\_exception import SSHException  
from netmiko.ssh\_exception import AuthenticationException

"Ввод аутентификационных данных в зашифрованном виде"  
username = input("Enter your SSH username: ")  
password = getpass()

"Открытие файла с командами и считывание их в переменную"  
with open("commands\_file") as f:  
    commands\_list = f.read().splitlines()

"Открытие файла со списком устройств и считывание их в переменную"  
with open("devices\_file") as f:  
    devices\_list = f.read().splitlines()

"Подключение к устройству из заданного списка в цикле FOR"  
for devices in devices\_list:  
    print ("Connecting to device" " + devices)  
    ip\_address\_of\_device = devices  
    ios\_device = {  
        "device\_type": "cisco\_ios",  
        "ip": ip\_address\_of\_device,   
        "username": username,  
        "password": password  
    }

"Попытка обработать ошибку"   
    try:  
        net\_connect = ConnectHandler(\*\*ios\_device)

    #Срабатывание исключений в случае неудачного создания соединения      
    except (AuthenticationException):  
        print ("Authentication failure: " + ip\_address\_of\_device)  
        continue  
    except (NetMikoTimeoutException):  
        print ("Timeout to device: " + ip\_address\_of\_device)  
        continue  
    except (EOFError):  
        print ("End of file while attempting device " + ip\_address\_of\_device)  
        continue  
    except (SSHException):  
        print ("SSH Issue. Are you sure SSH is enabled? " + ip\_address\_of\_device)  
        continue  
    except Exception as unknown\_error:  
        print ("Some other error: " + str(unknown\_error))  
        continue  
    output = net\_connect.send\_config\_set(commands\_list)  
    print (output)

Таким образом, в случае неудачного SSH-подключения, наш скрипт попробует обработать ошибку согласно заданной логике "except" и вывести на экран информацию. Введем заведомо неправильные значения логин/пароля и выведем на экран результат:

[Изображение выглядит как текст

Автоматически созданное описание](https://nag.ru/upload/images/20210309-0007.png)

Для удобства показали как скрипт отработает в интерпретаторе с информацией о безуспешном входе на конкретное устройство. При этом дальнейшие подключения в цикле продолжили бы выполняться.

Проверка версии обеспечения

В современной сетевой инфраструктуре размещено большое количество различных коммутирующих и маршрутизирующих устройств, все они могут быть от различных вендоров со своими сетевыми ОС. Поэтому перед автоматической отправкой конфигурации, необходимо каким-то образом проходить валидацию, иначе команды попросту не будут приняты устройством, которому они не предназначались.

Соответственно формируется задача: даны 3 различных конфигурационных файла, которые нужно отправить соответствующему оборудованию, с выводом на экран информации об отправке.

"Импорт необходимых библиотек"  
from getpass import getpass  
from netmiko import ConnectHandler  
from netmiko.ssh\_exception import NetMikoTimeoutException  
from paramiko.ssh\_exception import SSHException  
from netmiko.ssh\_exception import AuthenticationException

"Ввод данных для подключения"  
username = input("Enter your SSH username: ")  
password = getpass()

"Считывание конфигураций в различные переменные"  
with open("commands\_file\_switch") as f:  
    commands\_list\_switch = f.read().splitlines()  
with open("commands\_file\_router") as f:  
    commands\_list\_router = f.read().splitlines()  
with open("commands\_file\_phyrouter") as f:  
    commands\_list\_phyrouter = f.read().splitlines()

"Считывание информации о сетевых устройствах"  
with open("devices\_file") as f:  
    devices\_list = f.read().splitlines()

"Запуск цикла с последовательным подключением к устройству"  
for devices in devices\_list:  
    print ("Connecting to device" " + devices) #Вывод на экран  
    ip\_address\_of\_device = devices

    "Формирования словаря для подключения"  
        ios\_device = {  
        "device\_type": "cisco\_ios",  
        "ip": ip\_address\_of\_device,   
        "username": username,  
        "password": password  
    }

        "Обработка исключений"  
        try:  
        net\_connect = ConnectHandler(\*\*ios\_device)  
    except (AuthenticationException):  
        print ("Authentication failure: " + ip\_address\_of\_device)  
        continue  
    except (NetMikoTimeoutException):  
        print ("Timeout to device: " + ip\_address\_of\_device)  
        continue  
    except (EOFError):  
        print ("End of file while attempting device " + ip\_address\_of\_device)  
        continue  
    except (SSHException):  
        print ("SSH Issue. Are you sure SSH is enabled? " + ip\_address\_of\_device)  
        continue  
    except Exception as unknown\_error:  
        print ("Some other error: " + str(unknown\_error))  
        continue

    # Формирование списка сетевых ОС"  
    list\_versions = ["vios\_l2-ADVENTERPRISEK9-M",   
                     "VIOS-ADVENTERPRISEK9-M",  
                     "C1900-UNIVERSALK9-M",  
                     "C3750-ADVIPSERVICESK9-M"  
                     ]

    # Цикл по проверке сетевой ОС  
    for software\_ver in list\_versions:  
        print ("Checking for " + software\_ver)  
        output\_version = net\_connect.send\_command("show version")  
        int\_version = 0 # Сброс переменной для цикла  
        int\_version = output\_version.find(software\_ver) # Проверка ОС на сетевом устройстве

                # Условие о выводе информации по текущей ОС на устройстве  
        if int\_version > 0:  
            print ("Software version found: " + software\_ver)  
            break  
        else:  
            print ("Did not find " + software\_ver)

    "Если устройство одной из версий, то отправить соответствующий конфигурационный файл"   
        if software\_ver == "vios\_l2-ADVENTERPRISEK9-M":  
        print ("Running " + software\_ver + " commands")  
        output = net\_connect.send\_config\_set(commands\_list\_switch)  
    elif software\_ver == "VIOS-ADVENTERPRISEK9-M":  
        print ("Running " + software\_ver + " commands")  
        output = net\_connect.send\_config\_set(commands\_list\_router)  
    elif software\_ver == "C1900-UNIVERSALK9-M":  
        print ("Running " + software\_ver + " commands")  
        output =      net\_connect.send\_config\_set(commands\_list\_ph)  
    elif software\_ver == "C3750-ADVIPSERVICESK9-M":  
        print ("Running " + software\_ver + " commands")  
        output = net\_connect.send\_config\_set(commands\_list\_switch)    
    print (output)

Результат отработки представлен ниже:

[Изображение выглядит как текст

Автоматически созданное описание](https://nag.ru/upload/images/20210309-0009.png)

Скрипт подключился к устройству, взяв данные для SSH-сессии из файла, определил используемую ОС и отправил конфигурационный файл согласно заданному условию. Также в нем использовались уже изученные обработки исключений, которые позволили не прерываться процессу выполнения программы в случае ошибки, а продолжить свою работу.

Бонусный кейс с подключением без пароля

Подключение к сетевому оборудованию посредством аутентификации с помощью логина/пароля не всегда удобно и безопасно, даже если мы используем модуль getpass(). Netmiko дружит с SSH-ключами и на простом примере, мы покажем как сделать такое подключение.

Но для начала очень кратко по теории :

SSH-ключи представляют собой пару - закрытый и открытый ключ. Закрытый должен храниться в закрытом доступе у клиента, открытый отправляется на сервер и размещается в файле authorized\_keys.

В нашем случае при запуске скрипта, он должен будет получить открытый ключ и авторизоваться за счет сравнения с закрытым ключом на стороне сетевого оборудования.

Процесс генерации ключей уже много лет всем известен, если тема для вас новая, то предлагаю ознакомиться по [ссылке](https://firstvds.ru/technology/dobavit-ssh-klyuch).

Мы же перейдем непосредственно к практическому примеру с комментариями:

# Импорт библиотеки Netmiko  
from netmiko import ConnectHandler

# Объявление переменной с путем до открытого SSH-ключа сервера  
key\_file = "~/.ssh/test\_rsa"

# Объявление словаря с указанием типа аутентификации через ключ  
cisco1 = {  
    "device\_type": "cisco\_ios",  
    "host": "cisco1.lasthop.io",  
    "username": "testuser",  
    "use\_keys": True,  
    "key\_file": key\_file,  
}

# Создание подключения  
with ConnectHandler(\*\*cisco1) as net\_connect:  
    output = net\_connect.send\_command("show ip arp")

# Вывод на экран результата  
print(f"\n{output}\n")