PYTHON ДЛЯ СЕТЕВЫХ ИНЖЕНЕРОВ

ПОДКЛЮЧЕНИЕ К ОБОРУДОВАНИЮ

ВВОД ПАРОЛЯ

ВВОД ПАРОЛЯ

При подключении к оборудованию вручную, как правило, пароль также вводится вручную.

При автоматизизации подключения, надо решить каким образом будет передаваться пароль:

- запрашивать пароль при старте скрипта и считывать ввод пользователя
  - минус в том, что будет видно какие символы вводит пользователь
- записывать логин и пароль в каком-то файле
  - это не очень безопасно

МОДУЛЬ GETPASS

Модуль getpass позволяет запрашивать пароль, не отображая вводимые символы:

```
In [1]: import getpass

In [2]: password = getpass.getpass()
Password:

In [3]: print(password)
testpass
```

ПЕРЕМЕННЫЕ ОКРУЖЕНИЯ

Еще один вариант хранения пароля (а можно и пользователя) - переменные окружения.

Например, таким образом логин и пароль записываются в переменные:

```
$ export SSH_USER=user
$ export SSH_PASSWORD=userpass
```

А затем, в Python, считываются значения в переменные в скрипте:

```
import os

USERNAME = os.environ.get('SSH_USER')
PASSWORD = os.environ.get('SSH_PASSWORD')
```

МОДУЛЬ REXREST

МОДУЛЬ PEXPECT

Модуль pexpect позволяет автоматизировать интерактивные подключения, такие как:

- telnet
- ssh
- ftp

Для начала, модуль pexpect нужно установить:

```
pip install pexpect
```

МОДУЛЬ REXPECT

Логика работы rexpect такая:

- запускается какая-то программа
- rexpect ожидает определенный вывод (приглашение, запрос пароля и подобное)
- получив вывод, он отправляет команды/данные
- последние два действия повторяются столько, сколько нужно

При этом, сам rexpect не реализует различные утилиты, а использует уже готовые.

МОДУЛЬ REXPECT

В rexpect есть два основных инструмента:

- функция run()
- класс spawn

PEXPECT.RUN()

Функция run() позволяет вызвать какую-то программу и вернуть её вывод.

```
In [1]: import pexpect

In [2]: output = pexpect.run('ls -ls')

In [3]: print(output)
b'total 44\r\n4 -rw-r--r-- 1 vagrant vagrant 3203 Jul 14 07:15 1_pexpect.py\r\n4 -rw-r--r-- 1 vagrant vagra

In [4]: print(output.decode('utf-8'))
total 44
4 -rw-r--r-- 1 vagrant vagrant 3203 Jul 14 07:15 1_pexpect.py
4 -rw-r--r-- 1 vagrant vagrant 3393 Jul 14 07:15 2_telnetlib.py
4 -rw-r--r-- 1 vagrant vagrant 3452 Jul 14 07:15 3_paramiko.py
```

PEXPECT.SPAWN

Класс spawn поддерживает больше возможностей. Он позволяет взаимодействовать с вызванной программой, отправляя данные и ожидая ответ.

```
s = pexpect.spawn('ssh user@10.1.1.1')

s.expect('Password:')
s.sendline('userpass')
s.expect('>')
```

PEXPECT.SPAWN

Например, таким образом можно инициировать соединение SSH:

```
In [5]: ssh = pexpect.spawn('ssh cisco@192.168.100.1')
```

После выполнения этой строки, подключение готово.

PEXPECT . EXPECT

В pexpect.expect как шаблон может использоваться:

- регулярное выражение
- EOF - этот шаблон позволяет среагировать на исключение EOF
- TIMEOUT - исключение timeout (по умолчанию значение timeout = 30 секунд)
- compiled re

PEXPECT.EXPECT

```
In [6]: ssh.expect('[Pp]assword')
Out[6]: 0
```

Обратите внимание как описана строка, которую ожидает pexpect: [Pp]assword. Это регулярное выражение, которое описывает строку password или Password. То есть, методу expect можно передавать регулярное выражение как аргумент.

PEXPECT.EXPECT

Метод expect вернул число 0 в результате работы. Это число указывает, что совпадение было найдено и что это элемент с индексом ноль. Индекс тут фигурирует из-за того, что expect можно передавать список строк.

Например, можно передать список с двумя элементами:

```
In [7]: ssh = pexpect.spawn('ssh cisco@192.168.100.1')

In [8]: ssh.expect(['password', 'Password'])
Out[8]: 1
```

PEXPECT.SENDLINE

Для отправки команд используется команда sendline:

```
In [9]: ssh.sendline('cisco')
Out[9]: 6
```

Команда sendline отправляет строку, автоматически добавляет к ней перевод строки на основе значения os.linesep, а затем возвращает число указывающее сколько байт было записано.

EXPECT-SENDLINE

Для того чтобы попасть в режим enable цикл expect-sendline повторяется:

```
In [10]: ssh.expect('[>#]')
Out[10]: 0

In [11]: ssh.sendline('enable')
Out[11]: 7

In [12]: ssh.expect('[Pp]assword')
Out[12]: 0

In [13]: ssh.sendline('cisco')
Out[13]: 6

In [14]: ssh.expect('[>#]')
Out[14]: 0
```

ОТПРАВКА КОМАНДЫ

Теперь можно отправлять команду:

```
In [15]: ssh.sendline('sh ip int br')
Out[15]: 13
```

После отправки команды, pexpect надо указать до какого момента считать вывод. Указываем, что считать надо до #:

```
In [16]: ssh.expect('#')
Out[16]: 0
```

ПОЛУЧЕНИЕ ВЫВОДА С BEFORE

Вывод команды находится в атрибуте before:

```
In [17]: ssh.before
Out[17]: b'sh ip int br\r\nInterface                  IP-Address      OK? Method Status                Proto
```

ПОЛУЧЕНИЕ ВЫВОДА С BEFORE

Так как результат выводится в виде последовательности байтов, надо конвертировать ее в строку:

```
In [18]: show_output = ssh.before.decode('utf-8')

In [19]: print(show_output)
sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.1   YES NVRAM  up                    up
Ethernet0/1                192.168.200.1   YES NVRAM  up                    up
Ethernet0/2                19.1.1.1        YES NVRAM  up                    up
Ethernet0/3                192.168.230.1   YES NVRAM  up                    up
Ethernet0/3.100            10.100.0.1      YES NVRAM  up                    up
Ethernet0/3.200            10.200.0.1      YES NVRAM  up                    up
Ethernet0/3.300            10.30.0.1       YES NVRAM  up                    up
R1
```

ЗАВЕРШЕНИЕ СЕССИИ

Завершается сессия вызовом метода close:

```
In [20]: ssh.close()

In [21]: ssh.closed
Out[21]: False
```

СПЕЦИАЛЬНЫЕ СИМВОЛЫ В SHELL

Pexpect не интерпретирует специальные символы shell, такие как >, |, *.

Для того, чтобы, например, команда ls -ls | grep SUMMARY отработала, нужно запустить shell таким образом:

```
In [1]: import pexpect

In [2]: p = pexpect.spawn('/bin/bash -c "ls -ls | grep pexpect"')

In [3]: p.expect(pexpect.EOF)
Out[3]: 0

In [4]: print(p.before)
b'4 -rw-r--r-- 1 vagrant vagrant 3203 Jul 14 07:15 1_pexpect.py\r\n'

In [5]: print(p.before.decode('utf-8'))
4 -rw-r--r-- 1 vagrant vagrant 3203 Jul 14 07:15 1_pexpect.py
```

PEXPECT.EOF

В предыдущем примере встретилось использование pexpect.EOF.

Это специальное значение, которое позволяет отреагировать на завершение исполнения команды или сессии, которая была запущена в spawn.

При вызове команды ls -ls pexpect не получает интерактивный сеанс. Команда выполняется и всё, на этом завершается её работа.

PEXPECT.EOF

Поэтому если запустить её и указать в expect приглашение, возникнет ошибка:

```
In [5]: p = pexpect.spawn('/bin/bash -c "ls -ls | grep SUMMARY"')

In [6]: p.expect('nattaur')
---------------------------------------------------------------------------
EOF                                       Traceback (most recent call last)
<ipython-input-9-9c71777698c2> in <module>()
----> 1 p.expect('nattaur')
...
```

Но, если передать в expect EOF, ошибки не будет.

МЕТОД PEXPECT.EXPECT

В pexpect.expect как шаблон может использоваться:

- регулярное выражение
- EOF - этот шаблон позволяет среагировать на исключение EOF
- TIMEOUT - исключение timeout (по умолчанию значение timeout = 30 секунд)
- compiled re

МЕТОД PEXPECT.EXPECT

Еще одна очень полезная возможность pexpect.expect: можно передавать не одно значение, а список:

```
In [7]: p = pexpect.spawn('/bin/bash -c "ls -ls | grep netmiko"')

In [8]: p.expect(['py3_convert', pexpect.TIMEOUT, pexpect.EOF])
Out[8]: 2
```

МЕТОД PEXPECT.EXPECT

Тут несколько важных моментов:

- когда pexpect.expect вызывается со списком, можно указывать разные ожидаемые строки
- кроме строк, можно указывать исключения
- pexpect.expect возвращает номер элемента списка, который сработал
  - в данном случае номер 2, так как исключение EOF находится в списке под номером два
- за счет такого формата можно делать ответвления в программе, в зависимости от того, с каким элементом было совпадение

ПРИМЕР ИСПОЛЬЗОВАНИЯ PEXPECT

Пример использования pexpect для подключения к оборудованию и передачи команды show (файл 1_pexpect.py):

```
import pexpect
import getpass
import sys

COMMAND = sys.argv[1]
USER = input("Username: ")
PASSWORD = getpass.getpass()
ENABLE_PASS = getpass.getpass(prompt='Enter enable password: ')

DEVICES_IP = ['192.168.100.1','192.168.100.2','192.168.100.3']
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ PEXPECT

Файл 1_pexpect.py:

```
for IP in DEVICES_IP:
    print("Connection to device {}".format( IP ))
    t = pexpect.spawn('ssh {}@{}'.format( USER, IP ))

    t.expect('Password:')
    t.sendline(PASSWORD)

    t.expect('>')
    t.sendline('enable')

    t.expect('Password:')
    t.sendline(ENABLE_PASS)

    t.expect('#')
    t.sendline("terminal length 0")

    t.expect('#')
    t.sendline(COMMAND)

    t.expect('#')
    print(t.before.decode('utf-8'))
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ PEXPECT

Выполнение скрипта выглядит так:

```
$ python 1_pexpect.py "sh ip int br"
Username: cisco
Password:
Enter enable secret:
Connection to device 192.168.100.1
sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.1   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.1.10.1       YES manual up                    up
FastEthernet0/1.20     10.1.20.1       YES manual up                    up
FastEthernet0/1.30     10.1.30.1       YES manual up                    up
FastEthernet0/1.40     10.1.40.1       YES manual up                    up
FastEthernet0/1.50     10.1.50.1       YES manual up                    up
FastEthernet0/1.60     10.1.60.1       YES manual up                    up
FastEthernet0/1.70     10.1.70.1       YES manual up                    up
R1
Connection to device 192.168.100.2
sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.2   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.2.10.1       YES manual up                    up
FastEthernet0/1.20     10.2.20.1       YES manual up                    up
FastEthernet0/1.30     10.2.30.1       YES manual up                    up
FastEthernet0/1.40     10.2.40.1       YES manual up                    up
FastEthernet0/1.50     10.2.50.1       YES manual up                    up
FastEthernet0/1.60     10.2.60.1       YES manual up                    up
FastEthernet0/1.70     10.2.70.1       YES manual up                    up
```

МОДУЛЬ TELNETLIB

TELNETLIB

Модуль telnetlib входит в стандартную библиотеку Python. Это реализация клиента telnet.

Принцип работы telnetlib напоминает pexpect, но есть несколько отличий. Самое заметное отличие в том, что telnetlib требует передачи байтовой строки, а не обычной.

ПОДКЛЮЧЕНИЕ

Подключение выполняется таким образом:

```
In [1]: telnet = telnetlib.Telnet('192.168.100.1')
```

МЕТОД READ_UNTIL

С помощью метода read_until указывается до какой строки считать вывод. При этом, как аргумент надо передавать не обычную строку, а байты:

```
In [2]: telnet.read_until(b'Username')
Out[2]: b'\r\n\r\nUser Access Verification\r\n\r\nUsername'
```

Метод read_until возвращает все, что он считал до указанной строки.

МЕТОД WRITE

Для передачи данных используется метод write. Ему нужно передавать байтовую строку:

```
In [3]: telnet.write(b'cisco\n')
```

ВВОД ПАРОЛЯ

Читаем вывод до слова Password и передаем пароль:

```
In [4]: telnet.read_until(b'Password')
Out[4]: b': cisco\r\nPassword'

In [5]: telnet.write(b'cisco\n')
```

ВВОД КОМАНДЫ

Теперь можно указать, что надо считать вывод до приглашения, а затем отправить команду:

```
In [6]: telnet.read_until(b'>')
Out[6]: b': \r\nR1>'

In [7]: telnet.write(b'sh ip int br\n')
```

ПОЛУЧЕНИЕ ВЫВОДА КОМАНДЫ С READ_UNTIL

После отправки команды можно продолжать использовать метод read_until:

```
In [8]: telnet.read_until(b'>')
Out[8]: b'sh ip int br\r\nInterface                  IP-Address      OK? Method Status                Protoc
```

ПОЛУЧЕНИЕ ВЫВОДА КОМАНДЫ С READ_VERY_EAGER

При использовании метода read_very_eager, можно отправить несколько команд, а затем считать весь доступный вывод:

```
In [9]: telnet.write(b'sh arp\n')

In [10]: telnet.write(b'sh clock\n')

In [11]: telnet.write(b'sh ip int br\n')

In [12]: all_result = telnet.read_very_eager().decode('utf-8')

In [13]: print(all_result)
sh arp
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  10.30.0.1                -   aabb.cc00.6530  ARPA   Ethernet0/3.300
Internet  10.100.0.1               -   aabb.cc00.6530  ARPA   Ethernet0/3.100
Internet  10.200.0.1               -   aabb.cc00.6530  ARPA   Ethernet0/3.200
Internet  19.1.1.1                 -   aabb.cc00.6520  ARPA   Ethernet0/2
Internet  192.168.100.1            -   aabb.cc00.6500  ARPA   Ethernet0/0
Internet  192.168.100.2          124   aabb.cc00.6600  ARPA   Ethernet0/0
Internet  192.168.100.3          143   aabb.cc00.6700  ARPA   Ethernet0/0
Internet  192.168.100.100        160   aabb.cc80.c900  ARPA   Ethernet0/0
Internet  192.168.200.1            -   0203.e800.6510  ARPA   Ethernet0/1
Internet  192.168.200.100         13   0800.27ac.16db  ARPA   Ethernet0/1
Internet  192.168.230.1            -   aabb.cc00.6530  ARPA   Ethernet0/3
R1>sh clock
*19:18:57.980 UTC Fri Nov 3 2017
R1>sh ip int br
Interface                  IP-Address      OK? Method Status                Protocol
```

ПОЛУЧЕНИЕ ВЫВОДА НЕСКОЛЬКИХ КОМАНД С READ_UNTIL

С read_until будет немного другой подход. Можно выполнить те же три команды, но затем получать вывод по одной за счет чтения до строки с приглашением:

```
In [14]: telnet.write(b'sh arp\n')

In [15]: telnet.write(b'sh clock\n')

In [16]: telnet.write(b'sh ip int br\n')

In [17]: telnet.read_until(b'>')
Out[17]: b'sh arp\r\nProtocol  Address          Age (min)  Hardware Addr   Type   Interface\r\nInternet  10.

In [18]: telnet.read_until(b'>')
Out[18]: b'sh clock\r\n*19:20:39.388 UTC Fri Nov 3 2017\r\nR1>'

In [19]: telnet.read_until(b'>')
Out[19]: b'sh ip int br\r\nInterface                  IP-Address      OK? Method Status                Proto
```

READ_UNTIL VS READ_VERY_EAGER

Важное отличие между read_until и read_very_eager заключается в том, как они реагируют на отсутствие вывода.

Метод read_until ждет определенную строку. По умолчанию, если ее нет, метод "зависнет". Опциональный параметр timeout позволяет указать сколько ждать нужную строку:

```
In [20]: telnet.read_until(b'>', timeout=5)
Out[20]: b''
```

Если за указанное время строка не появилась, возвращается пустая строка.

Метод read_very_eager просто вернет пустую строку, если вывода нет:

```
In [21]: telnet.read_very_eager()
Out[21]: b''
```

МЕТОД EXPECT

Метод expect позволяет указывать список с регулярными выражениями. Он работает похоже на pexpect, но в модуле telnetlib всегда надо передавать список регулярных выражений.

```
In [22]: telnet.write(b'sh clock\n')

In [23]: telnet.expect([b'[>#]'])
Out[23]:
(0,
 <_sre.SRE_Match object; span=(46, 47), match=b'>'>,
 b'sh clock\r\n*19:35:10.984 UTC Fri Nov 3 2017\r\nR1>')
```

МЕТОД EXPECT

Метод expect возвращает кортеж их трех элементов:

- индекс выражения, которое совпало
- объект Match
- байтовая строка, которая содержит все считанное до регулярного выражения и включая его

МЕТОД EXPECT

```
In [24]: telnet.write(b'sh clock\n')

In [25]: regex_idx, match, output = telnet.expect([b'[>#]'])

In [26]: regex_idx
Out[26]: 0

In [27]: match.group()
Out[27]: b'>'

In [28]: match
Out[28]: <_sre.SRE_Match object; span=(46, 47), match=b'>'>

In [29]: match.group()
Out[29]: b'>'

In [30]: output
Out[30]: b'sh clock\r\n*19:37:21.577 UTC Fri Nov 3 2017\r\nR1>'

In [31]: output.decode('utf-8')
Out[31]: 'sh clock\r\n*19:37:21.577 UTC Fri Nov 3 2017\r\nR1>'
```

ЗАКРЫТИЕ СОЕДИНЕНИЯ

Закрывается соединение методом close:

```
In [32]: telnet.close()
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ TELNETLIB

Файл 2_telnetlib.py:

```
import telnetlib
import time
import getpass
import sys

COMMAND = sys.argv[1].encode('utf-8')
USER = input("Username: ").encode('utf-8')
PASSWORD = getpass.getpass().encode('utf-8')
ENABLE_PASS = getpass.getpass(prompt='Enter enable password: ').encode('utf-8')

DEVICES_IP = ['192.168.100.1','192.168.100.2','192.168.100.3']
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ TELNETLIB

Файл 2_telnetlib.py:

```
for IP in DEVICES_IP:
    print("Connection to device {}".format( IP ))
    t = telnetlib.Telnet(IP)

    t.read_until(b"Username:")
    t.write(USER + b'\n')

    t.read_until(b"Password:")
    t.write(PASSWORD + b'\n')
    t.write(b"enable\n")

    t.read_until(b"Password:")
    t.write(ENABLE_PASS + b'\n')
    t.write(b"terminal length 0\n")
    t.write(COMMAND + b'\n')

    time.sleep(5)

    output = t.read_very_eager().decode('utf-8')
    print(output)
```

ПРИМЕР ИСПОЛЬЗОВАНИЯ TELNETLIB

Выполнение скрипта:

```
$ python 2_telnetlib.py "sh ip int br"
Username: cisco
Password:
Enter enable secret:
Connection to device 192.168.100.1

R1#terminal length 0
R1#sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.1   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.1.10.1       YES manual up                    up
FastEthernet0/1.20     10.1.20.1       YES manual up                    up
FastEthernet0/1.30     10.1.30.1       YES manual up                    up
FastEthernet0/1.40     10.1.40.1       YES manual up                    up
FastEthernet0/1.50     10.1.50.1       YES manual up                    up
FastEthernet0/1.60     10.1.60.1       YES manual up                    up
FastEthernet0/1.70     10.1.70.1       YES manual up                    up
R1#
Connection to device 192.168.100.2

R2#terminal length 0
R2#sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.2   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.2.10.1       YES manual up                    up
FastEthernet0/1.20     10.2.20.1       YES manual up                    up
FastEthernet0/1.30     10.2.30.1       YES manual up                    up
```

МОДУЛЬ PARAMIKO

МОДУЛЬ PARAMIKO

Paramiko это реализация протокола SSHv2 на Python. Paramiko предоставляет функциональность клиента и сервера.

Так как Paramiko не входит в стандартную библиотеку модулей Python, его нужно установить:

```
pip install paramiko
```

МОДУЛЬ PARAMIKO

Пример использования Paramiko (файл 3_paramiko.py):

```
import paramiko
import getpass
import sys
import time

COMMAND = sys.argv[1]
USER = input("Username: ")
PASSWORD = getpass.getpass()
ENABLE_PASS = getpass.getpass(prompt='Enter enable password: ')

DEVICES_IP = ['192.168.100.1','192.168.100.2','192.168.100.3']
```

МОДУЛЬ PARAMIKO

Пример использования Paramiko (файл 3_paramiko.py):

```
for IP in DEVICES_IP:
    print("Connection to device {}".format( IP ))
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

    client.connect(hostname=IP, username=USER, password=PASSWORD,
                   look_for_keys=False, allow_agent=False)
    ssh = client.invoke_shell()

    ssh.send("enable\n")
    ssh.send(ENABLE_PASS + '\n')
    time.sleep(1)

    ssh.send("terminal length 0\n")
    time.sleep(1)
    print(ssh.recv(1000).decode('utf-8'))

    ssh.send(COMMAND + "\n")
    time.sleep(2)
    result = ssh.recv(5000).decode('utf-8')
    print(result)
```

МОДУЛЬ PARAMIKO

- client = paramiko.SSHClient() - этот класс представляет соединение к SSH-серверу. Он выполняет аутентификацию клиента.
- client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
  - set_missing_host_key_policy - устанавливает какую политику использовать, когда выполнятся подключение к серверу, ключ которого неизвестен.
  - paramiko.AutoAddPolicy() - политика, которая автоматически добавляет новое имя хоста и ключ в локальный объект HostKeys.

МОДУЛЬ PARAMIKO

- client.connect - метод, который выполняет подключение к SSH-серверу и аутентифицирует подключение
  - hostname - имя хоста или IP-адрес
  - username - имя пользователя
  - password - пароль
  - look_for_keys - по умолчанию paramiko выполняет аутентификацию по ключам. Чтобы отключить это, надо поставить поставив False
  - allow_agent - paramiko может подключаться к локальному SSH агенту ОС. Это нужно при работе с ключами, а так как, в данном случае, аутентификация выполняется по логину/паролю, это нужно отключить.

МОДУЛЬ PARAMIKO

- ssh = client.invoke_shell() - после выполнения предыдущей команды уже есть подключение к серверу. Метод invoke_shell позволяет установить интерактивную сессию SSH с сервером.

МОДУЛЬ PARAMIKO

- Внутри установленной сессии выполняются команды и получаются данные:
  - ssh.send - отправляет указанную строку в сессию
  - ssh.recv - получает данные из сессии. В скобках указывается максимальное значение в байтах, которое можно получить. Этот метод возвращает считанную строку
- Кроме этого, между отправкой команды и считыванием, кое-где стоит строка time.sleep
  - с помощью неё указывается пауза - сколько времени подождать, прежде чем скрипт продолжит выполняться. Это делается для того, чтобы дождаться выполнения команды на оборудовании

МОДУЛЬ PARAMIKO

Так выглядит результат выполнения скрипта:

```
$ python 3_paramiko.py "sh ip int br"
Username: cisco
Password:
Enter enable secret:
Connection to device 192.168.100.1

R1>enable
Password:
R1#terminal length 0


R1#
sh ip int br
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.1   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.1.10.1       YES manual up                    up
FastEthernet0/1.20     10.1.20.1       YES manual up                    up
FastEthernet0/1.30     10.1.30.1       YES manual up                    up
FastEthernet0/1.40     10.1.40.1       YES manual up                    up
FastEthernet0/1.50     10.1.50.1       YES manual up                    up
FastEthernet0/1.60     10.1.60.1       YES manual up                    up
FastEthernet0/1.70     10.1.70.1       YES manual up                    up
R1#
Connection to device 192.168.100.2

R2>enable
Password:
R2#terminal length 0
```

МОДУЛЬ PARAMIKO

Обратите внимание, что в вывод попал и процесс ввода пароля enable и команда terminal length.

Это связано с тем, что paramiko собирает весь вывод в буфер. И, при вызове метода recv (например, ssh.recv(1000)), paramiko возвращает всё, что есть в буфере. После выполнения recv, буфер пуст.

МОДУЛЬ PARAMIKO

Поэтому, если нужно получить только вывод команды sh ip int br, то надо оставить recv, но не делать print:

```
ssh.send("enable\n")
ssh.send(ENABLE_PASS + '\n')
time.sleep(1)

ssh.send("terminal length 0\n")
time.sleep(1)
#Тут мы вызываем recv, но не выводим содержимое буфера
ssh.recv(1000)

ssh.send(COMMAND + "\n")
time.sleep(3)
result = ssh.recv(5000).decode('utf-8')
print(result)
```

МОДУЛЬ NETMIKO

МОДУЛЬ NETMIKO

Netmiko это модуль, который позволяет упростить использование paramiko для сетевых устройств.

Грубо говоря, netmiko это такая "обертка" для paramiko.

Сначала netmiko нужно установить:

```
pip install netmiko
```

МОДУЛЬ NETMIKO

Пример использования netmiko (файл 4_netmiko.py):

```
from netmiko import ConnectHandler
import getpass
import sys


COMMAND = sys.argv[1]
USER = input("Username: ")
PASSWORD = getpass.getpass()
ENABLE_PASS = getpass.getpass(prompt='Enter enable password: ')

DEVICES_IP = ['192.168.100.1','192.168.100.2','192.168.100.3']
```

МОДУЛЬ NETMIKO

Пример использования netmiko (файл 4_netmiko.py):

```
for IP in DEVICES_IP:
    print("Connection to device {}".format( IP ))
    DEVICE_PARAMS = {'device_type': 'cisco_ios',
                     'ip': IP,
                     'username':USER,
                     'password':PASSWORD,
                     'secret':ENABLE_PASS }

    ssh = ConnectHandler(**DEVICE_PARAMS)
    ssh.enable()

    result = ssh.send_command(COMMAND)
    print(result)
```

МОДУЛЬ NETMIKO

- DEVICE_PARAMS - это словарь, в котором указываются параметры устройства
  - device_type - это предопределенные значения, которые понимает netmiko
    - в данном случае, так как подключение выполняется к устройству с Cisco IOS, используется значение 'cisco_ios'

МОДУЛЬ NETMIKO

- ssh = ConnectHandler(**DEVICE_PARAMS) - устанавливается соединение с устройством, на основе параметров, которые находятся в словаре
- ssh.enable() - переход в режим enable
  - пароль передается автоматически
  - используется значение ключа secret, который указан в словаре DEVICE_PARAMS
- result = ssh.send_command(COMMAND) - отправка команды и получение вывода

В этом примере не передается команда terminal length, так как netmiko по умолчанию, выполняет эту команду.

МОДУЛЬ NETMIKO

Так выглядит результат выполнения скрипта:

```
$ python 4_netmiko.py "sh ip int br"
Username: cisco
Password:
Enter enable password:
Connection to device 192.168.100.1
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.1   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.1.10.1       YES manual up                    up
FastEthernet0/1.20     10.1.20.1       YES manual up                    up
FastEthernet0/1.30     10.1.30.1       YES manual up                    up
FastEthernet0/1.40     10.1.40.1       YES manual up                    up
FastEthernet0/1.50     10.1.50.1       YES manual up                    up
FastEthernet0/1.60     10.1.60.1       YES manual up                    up
FastEthernet0/1.70     10.1.70.1       YES manual up                    up
Connection to device 192.168.100.2
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.2   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES NVRAM  up                    up
FastEthernet0/1.10     10.2.10.1       YES manual up                    up
FastEthernet0/1.20     10.2.20.1       YES manual up                    up
FastEthernet0/1.30     10.2.30.1       YES manual up                    up
FastEthernet0/1.40     10.2.40.1       YES manual up                    up
FastEthernet0/1.50     10.2.50.1       YES manual up                    up
FastEthernet0/1.60     10.2.60.1       YES manual up                    up
FastEthernet0/1.70     10.2.70.1       YES manual up                    up
Connection to device 192.168.100.3
Interface              IP-Address      OK? Method Status                Protocol
FastEthernet0/0        192.168.100.3   YES NVRAM  up                    up
```

ПОДДЕРЖИВАЕМЫЕ ТИПЫ УСТРОЙСТВ

Netmiko поддерживает несколько типов устройств:

- Arista vEOS
- Cisco ASA
- Cisco IOS
- Cisco IOS-XR
- Cisco SG300
- HP Comware7
- HP ProCurve
- Juniper Junos
- Linux
- и другие

Актуальный список можно посмотреть в репозитории модуля.

СЛОВАРЬ, ОПРЕДЕЛЯЮЩИЙ ПАРАМЕТРЫ УСТРОЙСТВ

В словаре могут указываться такие параметры:

```
cisco_router = {'device_type': 'cisco_ios', # предопределенный тип устройства
                'ip': '192.168.1.1', # адрес устройства
                'username': 'user', # имя пользователя
                'password': 'userpass', # пароль пользователя
                'secret': 'enablepass', # пароль режима enable
                'port': 20022, # порт SSH, по умолчанию 22
                 }
```

ПОДКЛЮЧЕНИЕ ПО SSH

```
ssh = ConnectHandler(**cisco_router)
```

РЕЖИМ ENABLE

Перейти в режим enable:

```
ssh.enable()
```

Выйти из режима enable:

```
ssh.exit_enable_mode()
```

ОТПРАВКА КОМАНД

В netmiko есть несколько способов отправки команд:

- send_command - отправить одну команду
- send_config_set - отправить список команд
- send_config_from_file - отправить команды из файла (использует внутри метод send_config_set)
- send_command_timing - отправить команду и подождать вывод на основании таймера

SEND_COMMAND

Метод send_command позволяет отправить одну команду на устройство.

Например:

```
result = ssh.send_command("show ip int br")
```

SEND_COMMAND

Метод работает таким образом:

- отправляет команду на устройство и получает вывод до строки с приглашением или до указанной строки
  - приглашение определяется автоматически
  - если на вашем устройстве оно не определилось, можно просто указать строку, до которой считывать вывод
  - ранее так работал метод send_command_expect, но с версии 1.0.0 так работает send_command, а метод send_command_expect оставлен для совместимости
- метод возвращает вывод команды

SEND_COMMAND

- методу можно передавать такие параметры:
  - command_string - команда
  - expect_string - до какой строки считывать вывод
  - ```delay_factor - параметр позволяет увеличить задержку до начала поиска строки
  - max_loops - количество итераций, до того как метод выдаст ошибку (исключение). По умолчанию 500
  - strip_prompt - удалить приглашение из вывода. По умолчанию удаляется
  - strip_command - удалить саму команду из вывода

В большинстве случаев, достаточно будет указать только команду.

SEND_CONFIG_SET

Метод send_config_set позволяет отправить несколько команд конфигурационного режима.

Пример использования:

```
commands = ["router ospf 1",
            "network 10.0.0.0 0.255.255.255 area 0",
            "network 192.168.100.0 0.0.0.255 area 1"]

result = ssh.send_config_set(commands)
```

SEND_CONFIG_SET

Метод работает таким образом:

- заходит в конфигурационный режим,
- затем передает все команды
- и выходит из конфигурационного режима
  - в зависимости от типа устройства, выхода из конфигурационного режима может и не быть. Например, для IOS-XR выхода не будет, так как сначала надо закомитить изменения

SEND_CONFIG_FROM_FILE

Метод send_config_from_file отправляет команды из указанного файла в конфигурационный режим.

Пример использования:

```
result = ssh.send_config_from_file("config_ospf.txt")
```

Метод открывает файл, считывает команды и передает их методу send_config_set.

ДОПОЛНИТЕЛЬНЫЕ МЕТОДЫ

Кроме перечисленных методов для отправки команд, netmiko поддерживает такие методы:

- config_mode - перейти в режим конфигурации
  - ssh.config_mode()
- exit_config_mode - выйти из режима конфигурации
  - ssh.exit_config_mode()

ДОПОЛНИТЕЛЬНЫЕ МЕТОДЫ

- check_config_mode - проверить находится ли netmiko в режиме конфигурации (возвращает True, если в режиме конфигурации и False - если нет)
  - ssh.check_config_mode()
- find_prompt - возвращает текущее приглашение устройства
  - ssh.find_prompt()
- commit - выполнить commit на IOS-XR и Juniper
  - ssh.commit()
- disconnect - завершить соединение SSH

TELNET

С версии 1.0.0 netmiko поддерживает подключения по Telnet. Пока что, только для Cisco IOS устройств.

Внутри, netmiko использует telnetlib, для подключения по Telnet. Но, при этом, предоставляет тот же интерфейс дял работы, что и подключение по SSH.

TELNET

Для того, чтобы подключиться по Telnet, достаточно в словаре, который определяет параметры подключения, указать тип устройства 'cisco_ios_telnet':

```
DEVICE_PARAMS = {'device_type': 'cisco_ios_telnet',
                 'ip': IP,
                 'username':USER,
                 'password':PASSWORD,
                 'secret':ENABLE_PASS }
```

TELNET

В остальном, методы, которые применимы к SSH, применимы и к Telnet. Пример, аналогичный примеру с SSH (файл 4_netmiko_telnet.py):

```
from netmiko import ConnectHandler
import getpass
import sys
import time

COMMAND = sys.argv[1]
USER = input("Username: ")
PASSWORD = getpass.getpass()
ENABLE_PASS = getpass.getpass(prompt='Enter enable password: ')

DEVICES_IP = ['192.168.100.1','192.168.100.2','192.168.100.3']
```

TELNET

Файл 4_netmiko_telnet.py:

```
for IP in DEVICES_IP:
    print("Connection to device {}".format( IP ))
    DEVICE_PARAMS = {'device_type': 'cisco_ios_telnet',
                     'ip': IP,
                     'username':USER,
                     'password':PASSWORD,
                     'secret':ENABLE_PASS,
                     'verbose': True}
    ssh = ConnectHandler(**DEVICE_PARAMS)
    ssh.enable()

    result = ssh.send_command(COMMAND)
    print(result)
```

TELNET

Аналогично работают и методы:

- send_command_timing()
- find_prompt()
- send_config_set()
- send_config_from_file()
- check_enable_mode()
- disconnect()