Работа с файлами

В реальной жизни, для того чтобы полноценно использовать всё, что рассматривалось до этого раздела, надо разобраться как работать с файлами.

При работе с сетевым оборудованием (и не только), файлами могут быть:

- конфигурации (простые, не структурированные текстовые файлы)
  - работа с ними рассматривается в этом разделе
- шаблоны конфигураций
  - как правило, это какой-то специальный формат файлов.
  - в разделе Шаблоны конфигураций с Jinja рассматривается использование Jinja2 для создания шаблонов конфигураций
- файлы с параметрами подключений
  - как правило, это структурированные файлы, в каком-то определенном формате: YAML, JSON, CSV
  - в разделе Сериализация данных рассматривается как работать с такими файлами
- другие скрипты Python
  - в разделе Модули рассматривается как работать с модулями (другими скриптами Python)

В этом разделе рассматривается работа с простыми текстовыми файлами. Например, конфигурационный файл Cisco.

В работе с файлами есть несколько аспектов:

- открытие/закрытие
- чтение
- запись

> В этом разделе рассматривается только необходимый минимум для работы с файлами. Подробнее, в документации Python.

Открытие файлов

Для начала работы с файлом, его надо открыть.

`open()`

Для открытия файлов, чаще всего, используется функция `open()`:

```
file = open('file_name.txt', 'r')
```

В функции open():

- `'file_name.txt'` - имя файла
  - тут можно указывать не только имя, но и путь (абсолютный или относительный)
- `'r'` - режим открытия файла

Функция `open()` создает объект file, к которому потом можно применять различные методы, для работы с ним.

Режимы открытия файлов:

- `r` - открыть файл только для чтения (значение по умолчанию)
- `r+` - открыть файл для чтения и записи
- `w` - открыть файл для записи
  - если файл существует, то его содержимое удаляется
  - если файл не существует, то создается новый
- `w+` - открыть файл для чтения и записи
  - если файл существует, то его содержимое удаляется
  - если файл не существует, то создается новый
- `a` - открыть файл для дополнения записи. Данные добавляются в конец файла
- `a+` - открыть файл для чтения и записи. Данные добавляются в конец файла

> r - read; a - append; w - write

Чтение файлов

В Python есть несколько методов чтения файла:

- `read()` - считывает содержимое файла в строку
- `readline()` - считывает файл построчно
- `readlines()` - считывает строки файла и создает список из строк

Посмотрим как считывать содержимое файлов, на примере файла r1.txt:

```
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

`read()`

Метод `read()` - считывает весь файл в одну строку.

Пример использования метода `read()`:

```
In [1]: f = open('r1.txt')

In [2]: f.read()
Out[2]: '!\nservice timestamps debug datetime msec localtime show-timezone year\nservi
ce timestamps log datetime msec localtime show-timezone year\nservice password-encrypt
ion\nservice sequence-numbers\n!\nno ip domain lookup\n!\nip ssh version 2\n!\n'

In [3]: f.read()
Out[3]: ''
```

При повторном чтении файла в 3 строке, отображается пустая строка. Так происходит из-за того, что при вызове метода `read()`, считывается весь файл. И после того, как файл был считан, курсор остается в конце файла. Управлять положением курсора можно с помощью метода `seek()`.

`readline()`

Построчно файл можно считать с помощью метода `readline()`:

```
In [4]: f = open('r1.txt')

In [5]: f.readline()
Out[5]: '!\n'

In [6]: f.readline()
Out[6]: 'service timestamps debug datetime msec localtime show-timezone year\n'
```

Но, чаще всего, проще пройтись по объекту file в цикле, не используя методы `read...`:

```
In [7]: f = open('r1.txt')

In [8]: for line in f:
   ...:     print(line)
   ...:
!

service timestamps debug datetime msec localtime show-timezone year

service timestamps log datetime msec localtime show-timezone year

service password-encryption

service sequence-numbers

!

no ip domain lookup

!

ip ssh version 2

!
```

`readlines()`

Еще один полезный метод - `readlines()`. Он считывает строки файла в список:

```
In [9]: f = open('r1.txt')

In [10]: f.readlines()
Out[10]:
['!\n',
 'service timestamps debug datetime msec localtime show-timezone year\n',
 'service timestamps log datetime msec localtime show-timezone year\n',
 'service password-encryption\n',
 'service sequence-numbers\n',
 '!\n',
 'no ip domain lookup\n',
 '!\n',
 'ip ssh version 2\n',
 '!\n']
```

Если нужно получить строки файла, но без перевода строки в конце, можно воспользоваться методом `split` и как разделитель, указать символ `\n`:

```
In [11]: f = open('r1.txt')

In [12]: f.read().split('\n')
Out[12]:
['!',
 'service timestamps debug datetime msec localtime show-timezone year',
 'service timestamps log datetime msec localtime show-timezone year',
 'service password-encryption',
 'service sequence-numbers',
 '!',
 'no ip domain lookup',
 '!',
 'ip ssh version 2',
 '!',
 '']
```

Обратите внимание, что последний элемент списка - пустая строка.

Если перед выполнением `split()`, воспользоваться методом `rstrip()`, список будет без пустой строки в конце:

```
In [13]: f = open('r1.txt')

In [14]: f.read().rstrip().split('\n')
Out[14]:
['!',
 'service timestamps debug datetime msec localtime show-timezone year',
 'service timestamps log datetime msec localtime show-timezone year',
 'service password-encryption',
 'service sequence-numbers',
 '!',
 'no ip domain lookup',
 '!',
 'ip ssh version 2',
 '!']
```

`seek()`

До сих пор, файл каждый раз приходилось открывать заново, чтобы снова его считать. Так происходит из-за того, что после методов чтения, курсор находится в конце файла. И повторное чтение возвращает пустую строку.

Чтобы ещё раз считать информацию из файла, нужно воспользоваться методом `seek`, который перемещает курсор в необходимое положение.

Пример открытия файла и считывания содержимого:

```
In [15]: f = open('r1.txt')

In [16]: print(f.read())
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

Если вызывать ещё раз метод `read`, возвращается пустая строка:

```
In [17]: print(f.read())
```

Но, с помощью метода `seek`, можно перейти в начало файла (0 означает начало файла):

```
In [18]: f.seek(0)
```

После того, как, с помощью `seek`, курсор был переведен в начало файла, можно опять считывать содержимое:

```
In [19]: print(f.read())
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

Запись файлов

При записи, очень важно определиться с режимом открытия файла, чтобы случайно его не удалить:

- `w` - открыть файл для записи. Если файл существует, то его содержимое удаляется
- `a` - открыть файл для дополнения записи. Данные добавляются в конец файла

При этом, оба режима создают файл, если он не существует

Для записи в файл используются такие методы:

- `write()` - записать в файл одну строку
- `writelines()` - позволяет передавать в качестве аргумента список строк

`write()`

Метод `write` ожидает строку, для записи.

Для примера, возьмем список строк с конфигурацией:

```
In [1]: cfg_lines = ['!',
   ...:  'service timestamps debug datetime msec localtime show-timezone year',
   ...:  'service timestamps log datetime msec localtime show-timezone year',
   ...:  'service password-encryption',
   ...:  'service sequence-numbers',
   ...:  '!',
   ...:  'no ip domain lookup',
   ...:  '!',
   ...:  'ip ssh version 2',
   ...:  '!']
```

Открытие файла r2.txt в режиме для записи:

```
In [2]: f = open('r2.txt', 'w')
```

Преобразуем список команд в одну большую строку с помощью `join`:

```
In [3]: cfg_lines_as_string = '\n'.join(cfg_lines)

In [4]: cfg_lines_as_string
Out[4]: '!\nservice timestamps debug datetime msec localtime show-timezone year\nservi
ce timestamps log datetime msec localtime show-timezone year\nservice password-encrypt
ion\nservice sequence-numbers\n!\nno ip domain lookup\n!\nip ssh version 2\n!'
```

Запись строки в файл:

```
In [5]: f.write(cfg_lines_as_string)
```

Аналогично можно добавить строку вручную:

```
In [6]: f.write('\nhostname r2')
```

После завершения работы с файлом, его необходимо закрыть:

```
In [7]: f.close()
```

Так как ipython поддерживает команду cat, можно легко посмотреть содержимое файла:

```
In [8]: cat r2.txt
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
hostname r2
```

`writelines()`

Метод `writelines()` ожидает список строк, как аргумент.

Запись списка строк cfg_lines в файл:

```
In [1]: cfg_lines = ['!',
   ...:  'service timestamps debug datetime msec localtime show-timezone year',
   ...:  'service timestamps log datetime msec localtime show-timezone year',
   ...:  'service password-encryption',
   ...:  'service sequence-numbers',
   ...:  '!',
   ...:  'no ip domain lookup',
   ...:  '!',
   ...:  'ip ssh version 2',
   ...:  '!']

In [9]: f = open('r2.txt', 'w')

In [10]: f.writelines(cfg_lines)

In [11]: f.close()

In [12]: cat r2.txt
!service timestamps debug datetime msec localtime show-timezone yearservice timestamps
 log datetime msec localtime show-timezone yearservice password-encryptionservice sequ
ence-numbers!no ip domain lookup!ip ssh version 2!
```

В результате, все строки из списка, записались в одну строку файла, так как в конце строк не было символа `\n`.

Добавить перевод строки можно по-разному.
Например, можно просто обработать список в цикле:

```
In [13]: cfg_lines2 = []

In [14]: for line in cfg_lines:
   ....:     cfg_lines2.append( line + '\n' )
   ....:

In [15]: cfg_lines2
Out[15]:
['!\n',
 'service timestamps debug datetime msec localtime show-timezone year\n',
 'service timestamps log datetime msec localtime show-timezone year\n',
 'service password-encryption\n',
 'service sequence-numbers\n',
 '!\n',
 'no ip domain lookup\n',
 '!\n',
 'ip ssh version 2\n',
```

Или использовать list comprehensions:

```
In [16]: cfg_lines3 = [ line + '\n' for line in cfg_lines ]

In [17]: cfg_lines3
Out[17]:
['!\n',
 'service timestamps debug datetime msec localtime show-timezone year\n',
 'service timestamps log datetime msec localtime show-timezone year\n',
 'service password-encryption\n',
 'service sequence-numbers\n',
 '!\n',
 'no ip domain lookup\n',
 '!\n',
 'ip ssh version 2\n',
 '!\n']
```

Если любой, из получившихся списков записать заново в файл, то в нем уже будут переводы строк:

```
In [18]: f = open('r2.txt', 'w')

In [19]: f.writelines(cfg_lines3)

In [20]: f.close()

In [21]: cat r2.txt
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

Закрытие файлов

> В реальной жизни, для закрытия файлов, чаще всего, используется конструкция `with`. Её намного удобней использовать, чем закрывать файл явно. Но, так как в жизни можно встретить и метод `close`, в этом разделе рассматривается как его использовать.

После завершения работы с файлом, его нужно закрыть.
В некоторых случаях, Python может самостоятельно закрыть файл.
Но лучше на это не рассчитывать и закрывать файл явно.

`close()`

Метод close встречался в разделе запись файлов.
Там он был нужен для того, чтобы содержимое файла было записано на диск.

Для этого, в Python есть отдельный метод `flush()`.
Но, так как, в примере с записью файлов, не нужно было больше выполнять никаких операций, файл можно было закрыть.

Откроем файл r1.txt:

```
In [1]: f = open('r1.txt', 'r')
```

Теперь можно считать содержимое:

```
In [2]: print(f.read())
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

У объекта file есть специальный атрибут `closed`, который позволяет проверить закрыт файл или нет.
Если файл открыт, он возвращает `False`:

```
In [3]: f.closed
Out[3]: False
```

Теперь закрываем файл и снова проверяем `closed`:

```
In [4]: f.close()

In [5]: f.closed
Out[5]: True
```

Если попробовать прочитать файл, возникнет исключение:

```
In [6]: print(f.read())
---------------------------------------------------------------------
ValueError                          Traceback (most recent call last)
<ipython-input-53-2c962247edc5> in <module>()
----> 1 print(f.read())

ValueError: I/O operation on closed file
```

Использование `try/finally` для работы с файлами

С помощью обработки исключений, можно:

- перехватывать исключения, которые возникают, при попытке прочитать несуществующий файл
- закрывать файл, после всех операций, в блоке `finally`

Если попытаться открыть для чтения файл, которого не существует, возникнет такое исключение:

```
In [7]: f = open('r3.txt', 'r')
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-54-1a33581ca641> in <module>()
----> 1 f = open('r3.txt', 'r')

IOError: [Errno 2] No such file or directory: 'r3.txt'
```

С помощью конструкции `try/except`, можно перехватить это исключение и вывести своё сообщение:

```
In [8]: try:
  ....:     f = open('r3.txt', 'r')
  ....: except IOError:
  ....:     print('No such file')
  ....:
No such file
```

А с помощью части `finally`, можно закрыть файл, после всех операций:

```
In [9]: try:
  ....:     f = open('r1.txt', 'r')
  ....:     print(f.read())
  ....: except IOError:
  ....:     print('No such file')
  ....: finally:
  ....:     f.close()
  ....:
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!

In [10]: f.closed
Out[10]: True
```

Конструкция with

> Конструкция with называется менеджер контекста.

В Python существует более удобный способ работы с файлами, чем те, которые использовались до сих пор - конструкция `with`:

```
In [1]: with open('r1.txt', 'r') as f:
  ....:     for line in f:
  ....:         print(line)
  ....:
!

service timestamps debug datetime msec localtime show-timezone year

service timestamps log datetime msec localtime show-timezone year

service password-encryption

service sequence-numbers

!

no ip domain lookup

!

ip ssh version 2

!
```

Кроме того, конструкция `with` гарантирует закрытие файла автоматически.

Обратите внимание на то, как считываются строки файла:

```
for line in f:
    print(line)
```

Когда с файлом нужно работать построчно, лучше использовать такой вариант.

В предыдущем выводе, между строками файла были лишние пустые строки, так как print добавляет ещё один перевод строки.

Чтобы избавиться от этого, можно использовать метод `rstrip`:

```
In [2]: with open('r1.txt', 'r') as f:
  ....:     for line in f:
  ....:         print(line.rstrip())
  ....:
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!

In [3]: f.closed
Out[3]: True
```

И, конечно же, с конструкцией `with` можно использовать не только такой построчный вариант считывания, все методы, которые рассматривались до этого, также работают:

```
In [4]: with open('r1.txt', 'r') as f:
  ....:     print(f.read())
  ....:
!
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
!
no ip domain lookup
!
ip ssh version 2
!
```

Открытие двух файлов

Иногда нужно работать одновременно с двумя файлами. Например, надо записать некоторые строки из одного файла, в другой.

В таком случае, в блоке with можно открывать два файла таким образом:

```
In [5]: with open('r1.txt') as src, open('result.txt', 'w') as dest:
   ...:     for line in src:
   ...:         if line.startswith('service'):
   ...:             dest.write(line)
   ...:

In [6]: cat result.txt
service timestamps debug datetime msec localtime show-timezone year
service timestamps log datetime msec localtime show-timezone year
service password-encryption
service sequence-numbers
```

Это равнозначно таким двум блокам with:

```
In [7]: with open('r1.txt') as src:
   ...:     with open('result.txt', 'w') as dest:
   ...:         for line in src:
   ...:             if line.startswith('service'):
   ...:                 dest.write(line)
   ...:
```

Дополнительные материалы

Документация:

- Reading and Writing Files
- The with statement

Статьи:

- The Python "with" Statement by Example

Stackoverflow:

- What is the python “with” statement designed for?

Задания

Все задания и вспомогательные файлы можно скачать в репозитории.

> Если в заданиях раздела есть задания с буквами (например, 5.2a), то можно выполнить сначала задания без букв, а затем с буквами. Задания с буквами, как правило, немного сложнее заданий без букв и развивают/усложняют идею в соответствующем задании без буквы.
>
> Например, если в разделе есть задания: 5.1, 5.2, 5.2a, 5.2b, 5.3, 5.3a. Сначала, можно выполнить задания 5.1, 5.2, 5.3. А затем 5.2a, 5.2b, 5.3a.
>
> Однако, если задания с буквами получается сделать сразу, можно делать их по порядку.

Задание 6.1

Аналогично заданию 3.1 обработать строки из файла ospf.txt и вывести информацию по каждой в таком виде: Protocol: OSPF Prefix: 10.0.24.0/24 AD/Metric: 110/41 Next-Hop: 10.0.13.3 Last update: 3d18h Outbound Interface: FastEthernet0/0

Ограничение: Все задания надо выполнять используя только пройденные темы.

Задание 6.2

Создать скрипт, который будет обрабатывать конфигурационный файл config_sw1.txt:

- имя файла передается как аргумент скрипту

Скрипт должен возвращать на стандартный поток вывода команды из переданного конфигурационного файла, исключая строки, которые начинаются с '!'.

Между строками не должно быть дополнительного символа перевода строки.

Ограничение: Все задания надо выполнять используя только пройденные темы.

Задание 6.2a

Сделать копию скрипта задания 6.2.

Дополнить скрипт:

- Скрипт не должен выводить команды, в которых содержатся слова, которые указаны в списке ignore.

Ограничение: Все задания надо выполнять используя только пройденные темы.

```
ignore = ['duplex', 'alias', 'Current configuration']
```

Задание 6.2b

Дополнить скрипт из задания 6.2a:

- вместо вывода на стандартный поток вывода, скрипт должен записать полученные строки в файл config_sw1_cleared.txt

При этом, должны быть отфильтрованы строки, которые содержатся в списке ignore.

Строки, которые начинаются на '!' отфильтровывать не нужно.

Ограничение: Все задания надо выполнять используя только пройденные темы.

```
ignore = ['duplex', 'alias', 'Current configuration']
```

Задание 6.2c

Переделать скрипт из задания 6.2b:

- передавать как аргументы скрипту:
  - имя исходного файла конфигурации
  - имя итогового файла конфигурации

Внутри, скрипт должен отфильтровать те строки, в исходном файле конфигурации, в которых содержатся слова из списка ignore. И затем записать оставшиеся строки в итоговый файл.

Проверить работу скрипта на примере файла config_sw1.txt.

Ограничение: Все задания надо выполнять используя только пройденные темы.

```
ignore = ['duplex', 'alias', 'Current configuration']
```

Задание 6.3

Скрипт должен обрабатывать записи в файле CAM_table.txt таким образом чтобы:

- считывались только строки, в которых указаны MAC-адреса
- каждая строка, где есть MAC-адрес, должна обрабатываться таким образом, чтобы на стандартный поток вывода была выведена таблица вида:

```
100    01bb.c580.7000   Gi0/1
200    0a4b.c380.7010   Gi0/2
300    a2ab.c5a0.2000   Gi0/3
100    0a1b.1c80.7300   Gi0/4
500    02b1.3c80.7000   Gi0/5
200    1a4b.c580.5000   Gi0/6
300    0a1b.5c80.9010   Gi0/7
```

Ограничение: Все задания надо выполнять используя только пройденные темы.

Задание 6.3a

Сделать копию скрипта задания 6.3

Дополнить скрипт:

- Отсортировать вывод по номеру VLAN

Ограничение: Все задания надо выполнять используя только пройденные темы.

Задание 6.3b

Сделать копию скрипта задания 6.3a

Дополнить скрипт:

- Запросить у пользователя ввод номера VLAN.
- Выводить информацию только по указанному VLAN.

Ограничение: Все задания надо выполнять используя только пройденные темы.