Урок schedule

Работа с командной строкой (скрипты, аргументы). Периодические задачи (модуль schedule)

Аннотация

В уроке мы систематизируем уже накопленные к этому времени знания о командной строке как самом простом, с точки зрения реализации, интерфейсе пользователя, и о создании скриптов для командной строки на языке Python. Разбирается простой способ работы с параметрами командной строкой через список sys.argv из модуля sys.

1. Про командную строку

Большинство современных пользователей ПК знают, что такое графический интерфейс GUI, как выглядят окна, кнопки, поля для ввода данных и другие визуальные элементы. Такие элементы интерфейсов сейчас принято называть **виджетами**. Мы уже достаточно близко познакомились с ними в блоках PyQt и PyGame.

На заре эволюции пользовательских интерфейсов самым популярным был текстовый интерфейс — **командная строка**. Отчасти его популярность была связана с тем, что графические устройства были дорогими и *медленными*, но в современном мире дела уже обстоят не так. Графические платы и мониторы существенно

улучшили свои показатели и характеристики как по производительности, так и по цене.

Однако интерфейс командной строки, несмотря на свой архаизм, до сих пор «жив-здоров» и находит применение в различных сферах IT-индустрии, например, в серверном программировании и создании веб-приложений.

Давайте разберемся, почему так происходит и зачем IT-инженеру надо обязательно уметь работать с командной строкой?

Какие же преимущества имеет командная строка перед GUI?

1. Простота удаленной работы. Легче синхронизировать два экрана разного разрешения, которые выводят текст, чем два экрана, выводящих графическую информацию. Кроме того, для вывода графической информации надо передать существенно больший объем данных, а это зачастую требует хорошей сетевой инфраструктуры
2. Самим протоколом командной строки предусмотрены механизмы автоматизации, средства общения между командами и вывод ошибок. Проще говоря, если разработчик программы с графическим интерфейсом не заложил в нее возможности повторять на экране действия оператора или взаимодействовать с другой программой, то сделать это при необходимости будет затруднительно
3. Интерфейс командной строки уже поддерживает множество отлаженных временем команд и технологий
4. На большинстве серверов графическая оболочка отсутствует, чтобы не тратить лишние серверные ресурсы и не нагружать дополнительно сеть при удаленном подключении

Давайте попробуем немного разобраться с командной строкой и понять, как же она работает.

2. Виртуальные машины и WSL

Очень часто программисту или IT-инженеру, работающему в какой-либо операционной системе, нужно смоделировать работу другой операционной системы. Например, вы работаете на компьютере под управлением операционной системы Windows, а хотите проверить работу программы под операционной системой Linux. Конечно, можно поставить вторую операционную систему на свой компьютер, но это требует определенной квалификации и при ошибке может привести к выходу из строя основной ОС. Безопаснее установить **виртуальную машину**. Благодаря ей вы сможете получить *операционную систему как бы внутри другой операционной системы*.

Самое главное, что таких дочерних операционных систем может быть сколько угодно много. Их количество ограничено лишь ресурсами вашего компьютера.

В сети Интернет можно найти много готовых образов операционных систем для виртуальных машин. Мы же приготовили вам свою, но вы можете использовать и другие.

Что касается ПО виртуальных машин, то мы рекомендуем использовать **VirtualBox**, хотя на рынке существует множество аналогичных программ.

По **ссылке** вы можете скачать готовые образы ОС Ubuntu 17.04 и ОС Lite Linux для VirtualBox. (Это — zip-архив, и он немаленький). Чтобы их использовать, скачайте последнюю версию VirtualBox для вашей операционной системы.

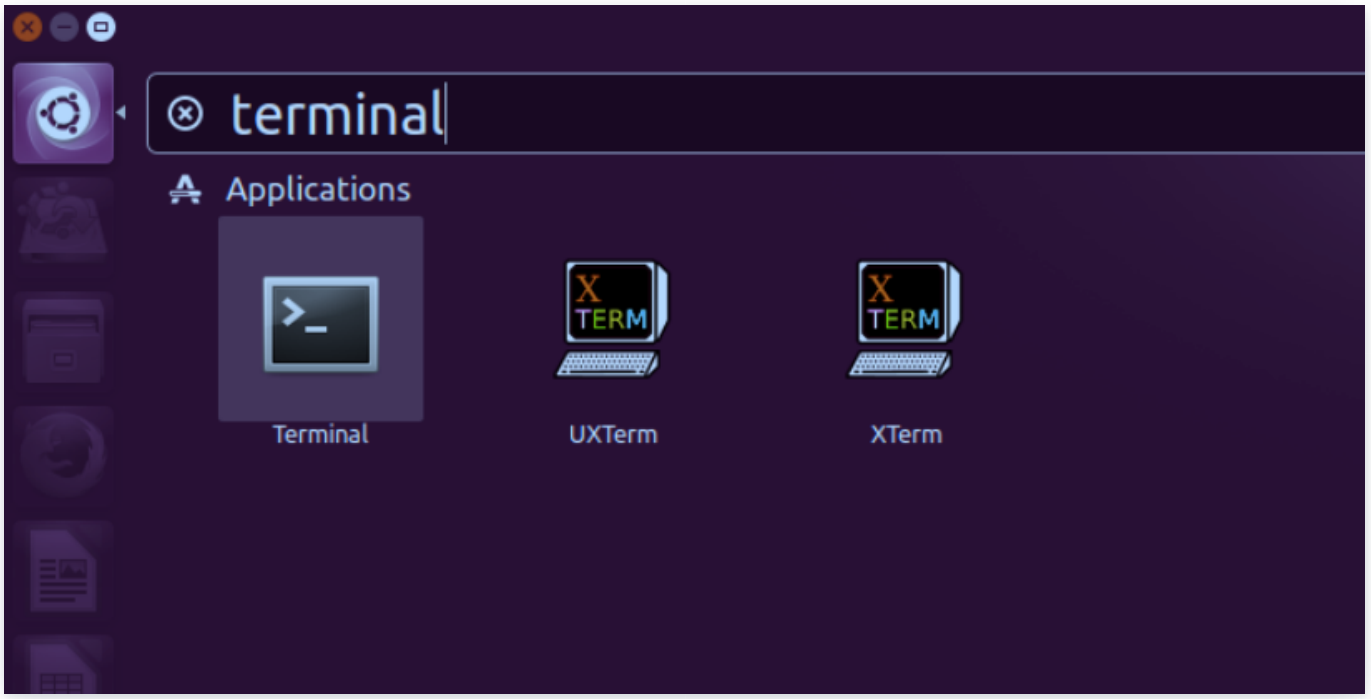
Для входа в систему (при появлении запроса) используйте логин: user и пароль: user.

А в **этом видео** показывается, как подключить готовый образ к VirtualBox.

Кроме того, Windows 10 дает возможность использовать Linux с использованием слоя совместимости для запуска Linux-приложений (**WSL**). В таком случае дистрибутив Linux можно установить непосредственно в Windows и работать в нем через командную строку. Инструкцию можно найти **здесь**. Так же в ОС Windows можно использовать в качестве терминала программу Git bash, тогда команды будут те же, что и для ОС Linux. Если у вас уже установлен git для Windows, то и git bash тоже установлен.

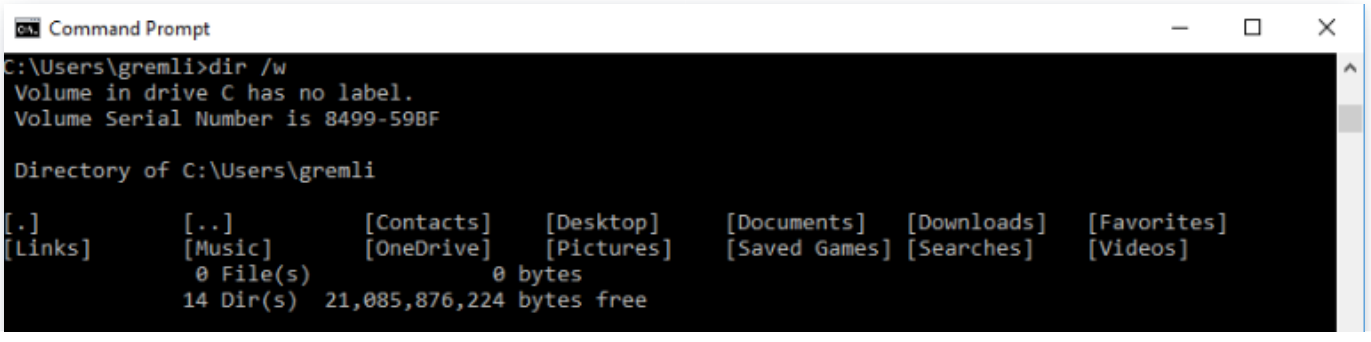
3. Запуск командной строки

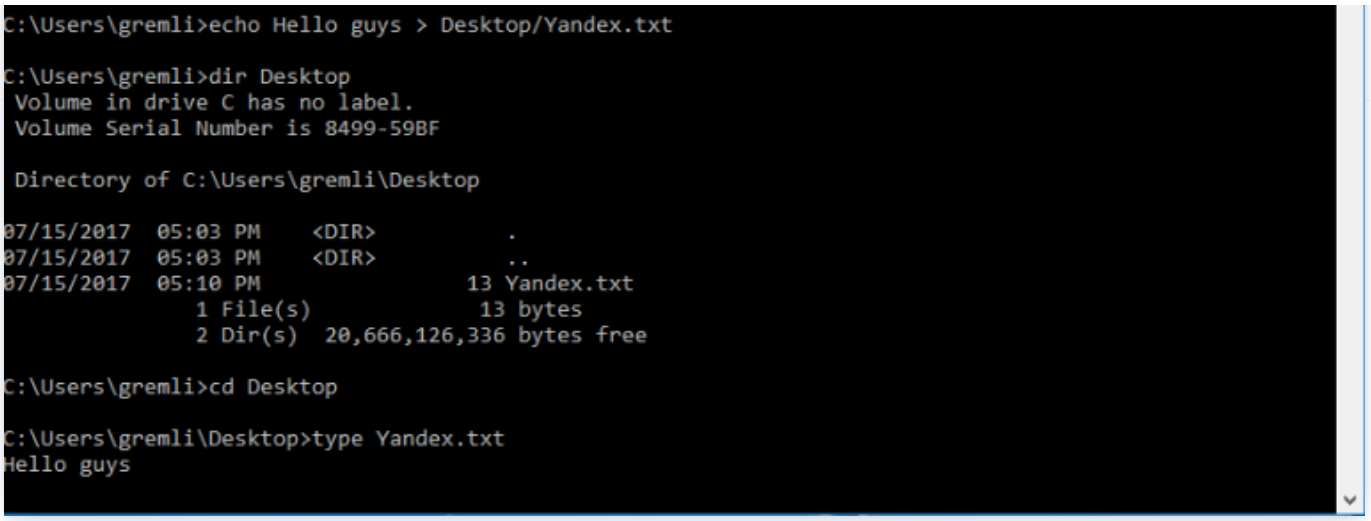
В разных UNIX-системах запуск командной строки осуществляется по-разному, но почти наверняка вам поможет иконка и ключевое слово **терминал** (terminal).



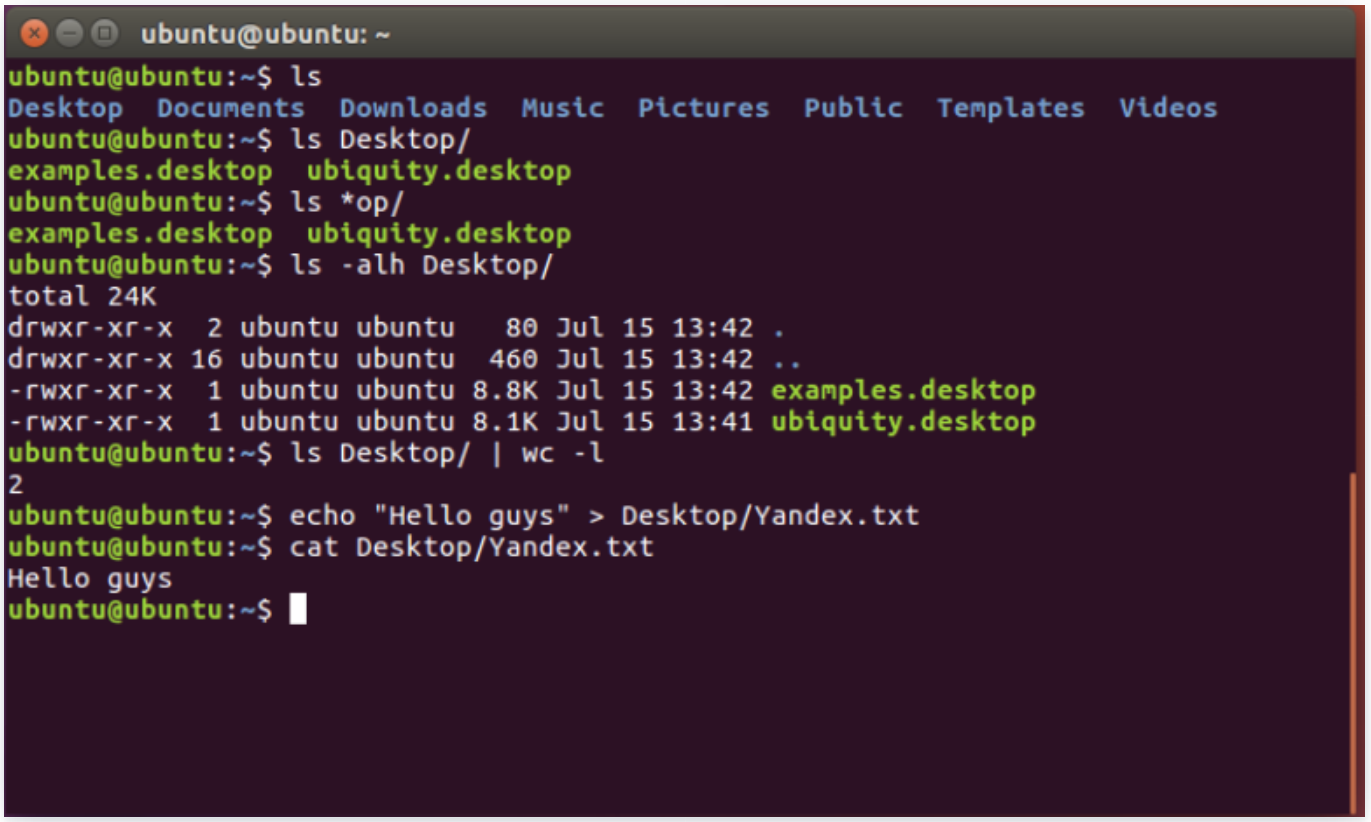
В каждой ОС есть командная строка. Даже если вы ее ни разу не видели, можете быть уверены, что и в вашей системе она есть. Например, мы с вами начали ее использовать еще на первом курсе для установки библиотек, для преобразования ui-файлов в классы Python и т. д. (На самом деле вкладка python shell в Wing IDE или панель в PyCharm, куда мы вводили значения от имени пользователей нашей программы с самого первого урока, являются надстройкой над командной строкой операционной системы с запущенным интерпретатором Python).

Вот примеры нескольких простых команд в ОС Windows (для запуска командной строки в Windows Вам нужно нажать Win+R, в ввести команду cmd и нажать ОК; также можно открыть командную строку непосредственно внутри PyCharm комбинацией клавиш — Alt+F12):





Для ОС Linux «картинка» будет слегка иной, но принципы все равно сохраняются. Посмотрите сами:



4. Принципы работы с командной строкой

После запуска программы командной строки (ее еще называют терминалом, консолью или shell) на экране появляется окно (обычно черное) с приглашением ввода. Это командная оболочка, которая будет выполнять наши команды. В данный момент операционная система ждет от пользователя ввода команд.

Наиболее популярная командная оболочка ОС семейства UNIX, которую вы, скорее всего, увидите, называется bash. У операционной системы Windows такой оболочкой является командная строка или cmd. В последние версии Windows внедряется новая командная оболочка, которая называется Powershell, которая хоть и поддерживает классический синтаксис cmd, но с некоторыми оговорками.

Давайте рассмотрим команду вывода содержимого директории `ls -alh Desktop/`. Результат ее работы вы можете увидеть в примере выше. Аналогичной по сути командой в ОС Windows является команда `dir`.

Она состоит из названия самой команды `ls` и **аргументов** (или параметров), передаваемых в командной строке.

В нашем примере аргументами является директория Desktop и **опции** (их называют также флагами или ключами) `-a -l -h`.

Все они влияют на результат работы команды. Например, Desktop указывает на папку, содержимое которой требуется вывести, но конкретную директорию можно и не передавать, тогда команда `ls` выведет содержимое текущей директории.

С помощью опции `-a, --all` можно вывести полный список файлов (включая скрытые и начинающиеся с «.»), опция `-l` выводит информацию о каждом файле на отдельной строке вместе с мета-информацией (владелец, права доступа, размер, время обновления), а опция `-h, --human-readable` приводит размеры файлов к более понятным для человека.

Есть еще одна разновидность аргументов — именованные аргументы. Например, когда информацию о каком-то из файлов выводить не нужно, это можно сделать так `ls --ignore=Desktop` или даже короче `ls -I Desktop`. Так мы смогли показать, что Desktop — это элемент, который нас не интересует.

Аргументы могут быть обязательными и не обязательными. Команда `ls`, например, не требует обязательных аргументов (она может быть запущена вообще без них), но так бывает не всегда. При удалении файла обязательно надо указать его имя.

Чтобы выполнить команду, надо знать ее имя. В большинстве случае команда — это файл. Операционной системе надо найти его и уже потом выполнить. Поэтому у системы есть список адресов, по которым она будет искать файл с указанным именем. Хранится этот список в специальной переменной PATH. Она называется *переменной окружения*.

Мы уже пользовались этой возможностью: при установке Python мы указывали, что исполняемый файл интерпретатора, а также утилиту pip надо добавить в PATH, чтобы использовать короткое название для их вызова, а не писать полный путь. Поэтому мы пишем:

```
pip install pillow
```

а не:

```
путь/к/pip/pip.exe install pillow
```

Команда `echo` позволяет вывести на экран любую информацию, в том числе и значения переменных.

С помощью `whereis` (`where` для Windows) можно узнать, где в файловой системе располагается нужная нам команда.

Ну и, конечно же, мы можем выполнить команду, указав полный путь к ее местонахождению.

5. Базовые команды

Приведем список самых важных и часто используемых команд, которые стоит запомнить (в скобочках приведем аналогичные команды ОС Windows):

1. `ls [-al] <директория> (dir [-al] <директория>)` — вывод списка файлов в директории (мы уже рассмотрели)
2. `man <команда> (help <команда>)` — документация по командам. Навигация в ней осуществляется с помощью кнопок вверх/вниз (j/k или Page Up/Page Down), а выход — с помощью кнопки q
3. `cd <директория>` — смена каталога (от англ. Change Directory). Директорию можно указывать относительно корня (cd /tmp, cd c:\temp), относительно текущего каталога (cd dir_name), а можно вообще опустить название. В этом случае переход произойдет в домашний каталог
4. `mkdir <директория>` — создание новой директории (от англ. Make Directory)
5. `cp [-r] <источник> <пункт назначения> (copy)` — копирование файла/директории (для копирования директорий необходимо использовать опцию `-r`)
6. `mv <источник> <пункт назначения> (move)` — перемещение файла/директории. Работает так же, как `cp`, только после создания нового объекта старый удаляется
7. `rm [-rf] <объект>` — удаление файлов, каталогов (элементов файловой системы). Для удаления директории требуется флаг `-r, --recursive`
8. `cat <объект> (type)` — вывести содержимое файла на экран

Для работы с файловой системой можно использовать алиасы:

- `.` — текущий каталог
- `~` — домашняя директория
- `..` — каталог на уровень выше текущего

6. Потоки, exit-коды

Большинство программ, работающих под управлением современных операционных систем, могут принимать данные со **стандартного потока ввода** — stdin (мы уже с ним встречались) и выводить данные на **стандартный поток вывода** — stdout.

Кроме этих двух потоков существует еще и стандартный поток для вывода ошибок. Он называется stderr.

После завершения любой команды в bash или cmd возвращается **код ответа** (Exitcode).

Exitcode — это число, значение которого равно нулю, если команда (да и любая программа) выполнена успешно. Если же значение Exitcode больше нуля, это означает, что в процессе возникли ошибки.

Например, значение Exitcode используется, чтобы прервать выполнение последовательного набора зависящих друг от друга команд, если в процессе их работы что-то пошло не так.

Получить exitcode в командной строке можно с помощью команды `echo $?`.

С Exitcode мы тоже уже встречались: каждый раз после выполнения программы в PyCharm дописывается

строка: `Process finished with exit code код_выхода`

7. Как запустить свой скрипт?

Когда вы написали свою программу на языке Python и хотите ее выполнить в консоли (в терминале, в командной строке), то у вас есть два способа это сделать:

1. Запустить интерпретатор Python и в качестве параметра передать ему имя вашего скрипта

   Для ОС Linux:

   ```
   python3 test.py 1 2 3 4
   ```

   Для ОС Windows:

   ```
   python test.py 1 2 3 4
   ```

2. Добавить в начало файла заголовок `#!/путь/до/интерпретатора`, по которому shell (командная оболочка) поймет, какой интерпретатор нужно выбрать для данного скрипта. Но это еще не все. Надо сделать файл с программой исполняемым через команду `chmod+x <имя_файла>`. Однако стоит помнить, что такой способ работает только для ОС семейства Linux

8. Простой парсинг аргументов в Python

Настало время создать программу на языке Python, которая умела бы обрабатывать параметры командной строки, как это делают другие команды.

Для этого нам понадобится список `argv` из модуля sys. В него Python упаковывает все аргументы командной строки, полученные при вызове программы.

```
import sys

print("my name is {}".format(sys.argv[0]))
print("first arg is: '{}' and last arg is '{}'".format(
    sys.argv[1], sys.argv[-1]))
```

```
python3 files/cmd1.py a1 b2 c3 d4

my name is files/cmd1.py
first arg is: 'a1' and last arg is 'd4'
```

Обратите внимание, что в `argv[0]` хранится **полное** имя программы, а остальные элементы списка содержат уже переданные параметры.

Вот только, если нет аргументов, на которые вы рассчитывали, можно легко «поймать» исключение.

```
python3 files/cmd1.py
```

```
my name is files/cmd1.py
    ---------------------------------------------------------------------------

    IndexError                                Traceback (most recent call last)

    ~/Google Drive/ЯндексЛицей/Уроки/materials/additional/2 year/15/files/cmd1.py in <module>()
          2
          3 print("my name is {}".format(sys.argv[0]))
    ----> 4 print("first arg is: '{}' and last arg is '{}'".format(sys.argv[1], sys.argv[-1]))

    IndexError: list index out of range
```

Так что не забывайте проверять аргумент перед обращением к нему. И раз дело мы имеем со списком, то самым простым способом будет подсчитать количество элементов в начале скрипта с помощью `len(sys.argv)`.

```
import sys

print("my name is {}".format(sys.argv[0]))
if len(sys.argv) > 1:
    print("first arg is: '{}' and last arg is '{}'".format(
        sys.argv[1], sys.argv[-1]))
else:
    print("No arguments")
```

```
python3 files/cmd2.py

my name is files/cmd2.py
No arguments
```

Работа с параметрами командной строки через список `sys.argv` интуитивно понятна, поскольку сводится к простой обработке списка. Но как только у нас появляются необязательные аргументы или зависимые друг от друга аргументы, могут начаться сложности. Как с ними справиться, мы расскажем на следующем уроке, а пока познакомимся с модулем schedule.

9. Периодические задачи

Мы все время писали программы, которые выполняются один раз (или до закрытия формы пользователем). Но достаточно часто встречаются задачи, для решения которых программа должна выполняться периодически, с каким-то интервалом. Это могут быть задачи, связанные с получением новой информации со стороннего ресурса, задачи резервного копирования и т. д.

Эту задачу можно решать сторонними средствами: в Windows для этого есть планировщик задач, в Linux — очень гибкая и мощная утилита cron, но можно и обойтись средствами, которые предоставляет нам Python. Для создания периодических задач в Python есть несколько способов, мы рассмотрим лишь один из них, самый удобный на наш взгляд — библиотеку schedule. Этот модуль не входит в стандартную библиотеку и его надо

устанавливать дополнительно:

```
pip install schedule
```

Работа с модулем достаточно простая. Давайте рассмотрим простое приложение, которое будет запускать на выполнение функцию каждые пять секунд:

```
import datetime

import schedule

i = 1


def job():
    global i
    print(f"Запустился {i} раз")
    i += 1
    print(datetime.datetime.now())


schedule.every(5).seconds.do(job)

while True:
    schedule.run_pending()
```

```
Запустился 1 раз
2019-11-01 12:18:27.172611
Запустился 2 раз
2019-11-01 12:18:32.173117
Запустился 3 раз
2019-11-01 12:18:37.173148
```

Модуль можно настраивать достаточно гибко:

```
schedule.every(10).minutes.do(job)
#каждые 10 минут

schedule.every().hour.do(job)
# каждый час

schedule.every().day.at("10:30").do(job)
# каждый день в 10-30

schedule.every().monday.do(job)
# каждый понедельник
```

```
schedule.every(2).wednesday.at("13:15").do(job)
# Каждую вторую среду в 13-15

schedule.every().minute.at(":17").do(job)
# каждую минуту в 17 секунду
```

Кроме того, можно передавать параметры в ту функцию, которую мы хотим вызывать:

```
import schedule

i = 0


def greet(name):
    global i
    print('Hello', name)
    i += 1
    if i == 5:
        return schedule.CancelJob # Отменяем задачу после 5 запуска


schedule.every(1).to(3).seconds.do(greet, name='Yandex')
# Запускаем задачу в случайное время время от 1 до 3 секунд

while True:
    schedule.run_pending()
```