

[КАК СТАТЬ АВТОРОМ](#)

True Tech Days: прямая трансляция



alexxis

28 мар в 03:31

7 уровней построения интерфейсов командной строки на Python

Простой

6 мин

5.4K

Настройка Linux*, Python*

[Тutorial](#)[Из песочницы](#)[Перевод](#)Автор оригинала: <https://medium.com/@yangzhou1993>

Автор

Выполняйте свои скрипты Python, как команды bash

Написание скриптов Python для создания интерфейсов командной строки (CLI) — широко используемый метод для DevOps и бэкенд разработки.

Ключом к реализации CLI в Python является встроенный модуль `argparse`. Он предоставляет все необходимые функции и позволяет использовать скрипт Python в качестве команды `bash`.

В этой статье будут представлены некоторые важные моменты создания CLI с помощью Python на 7 уровнях сложности.

1. Запускаем базовый интерфейс командной строки с модулем `argparse`

Прежде всего, давайте создадим файл с именем `test.py` и сделаем простой парсер аргументов:

```
import argparse

parser = argparse.ArgumentParser()
parser.parse_args()
```

После того, как мы определили этот парсер, теперь мы можем получать аргументы из командной строки. Каждый экземпляр `ArgumentParser` содержит параметр `--help` (или



+6



54



9

```
$ python test.py --help
```

```
usage: test.py [-h]
```

```
optional arguments: -h, --help show this help message and exit
```

Однако из-за того, что мы не определили другие аргументы, следующий ввод вызовет ошибку:

```
$ python test.py 9
```

```
usage: test.py [-h]
```

```
test.py: error: unrecognized arguments: 9
```

2. Определяем позиционные аргументы

В командах `bash` есть два типа аргументов. Один позиционный, другой необязательный.

Например, команда `cp` имеет два позиционных аргумента:

```
$ cp source_file dest_file
```

`source_file` указывает источники копируемого файла. И аргумент `dest_file` указывает место для копирования файла.

Мы также можем определить позиционный аргумент для нашего интерфейса командной строки Python:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("num", type=int)
args = parser.parse_args()
print(args.num**2)
```

Как видно из приведенного выше кода, мы определили имя аргумента как `num` и указали его тип как `int`.

Теперь, если ввод из терминала является допустимым целым числом, наш скрипт Python будет печатать его квадрат:

```
$ python test.py 9
```

81

3. Добавляем больше полезной информации и описание

Как и многие команды Linux, хороший интерфейс командной строки должен предоставлять пользователям достаточно информации с помощью опции `--help`.

С этой целью мы можем указать специальную информацию для соответствующих функций модуля `argparse`.

```
import argparse

parser = argparse.ArgumentParser(prog='get_square',description='A CLI to calculate the square of an integer')
parser.add_argument("num", type=int, help='An integer that needs to get the square value')
args = parser.parse_args()
print(args.num**2)
```

Как показано выше, мы добавили информацию о программе и описании в экземпляр `ArgumentParser`, чтобы дать ему имя и подробное описание. Кроме того, мы дали функции `add_argument()` справочное сообщение, чтобы наши пользователи знали, для чего нужен этот аргумент.

Теперь давайте снова посмотрим на вывод опции `-h`:

```
$ python test.py -h
```

```
usage: get_square [-h] num
```

```
A CLI to calculate the square of an integer.
```

```
positional arguments:
```

```
num An integer that needs to get the square value.
```

```
optional arguments:
```

```
-h, --help show this help message and exit
```

Это удобнее и информативнее, не правда ли? :)

4. Определяем необязательные аргументы

Помимо позиционных аргументов, для многих интерфейсов командной строки также необходимы необязательные аргументы.

Встроенная опция `--help` всех экземпляров `ArgumentParser` является необязательным аргументом. Мы можем добавить еще одну к предыдущему примеру:

```
import argparse

parser = argparse.ArgumentParser(prog='get_square',description='A CLI to calculate the square of a number')
parser.add_argument("num", type=int, help='An integer that needs to get the square of')
parser.add_argument('--verbose',help='Print more info.')

args = parser.parse_args()

if args.verbose:
    print(f'The square of {args.num} is {args.num**2}')
else:
    print(args.num**2)
```

Вышеприведенная программа добавила необязательный аргумент с именем:

`--verbose`. Если для этого аргумента есть значение, программа распечатает более подробную информацию для расчета.

Теперь давайте попробуем его использовать:

```
$ python test.py 9 --verbose 1
```

```
The square of 9 is 81
```

```
$ python test.py 9 --verbose 90908
```

```
The square of 9 is 81
```

```
$ python test.py 9 --verbose
```

```
usage: get_square [-h] [--verbose VERBOSE] num
```

```
get_square: error: argument --verbose: expected one argument
```

Досадная проблема заключается в том, что мы должны указать значение для этой опции, чтобы она работала, независимо от того, какое это значение. Если значения вообще нет, как показано выше, появится сообщение об ошибке.

Это связано с тем, что `args.verbose` имеет значение `None`, если мы используем его напрямую, не присваивая ему никаких значений.

Это не так удобно, как другие команды Linux. Поэтому нам нужно оптимизировать его через параметры `action`.

5. Определяем специальные действия для необязательных аргументов

Давайте немного изменим эту строку кода:

```
parser.add_argument('--verbose', help='Print more info.', action='store_true')
```

Параметр `action` функции `add_argument()` может указать, как обрабатывать значения этой опции.

В нашем случае `store_true` означает, что если указан параметр, программа по умолчанию присвоит значение `True` для `args.verbose`.

А затем выполните скрипт с опцией `--verbose` напрямую:

```
$ python test.py 9 --verbose
```

```
The square of 9 is 81
```

Помимо `store_true` или `store_false`, Python также предоставляет другие варианты параметров `action`. Давайте попробуем несколько:

Используйте "store_const" action

Мы можем использовать параметр `store_const` в качестве действия и присвоить `const` необязательному аргументу:

```
parser.add_argument('--verbose', help='Print more info.', action='store_const', c
```

Поэтому значение `args.verbose` всегда будет равным 1.

Параметр `count` подсчитывает, сколько раз встречается необязательный аргумент. Давайте отредактируем наш код и попробуем:

```
import argparse

parser = argparse.ArgumentParser(prog='get_square',description='A CLI to calculate the square of a number')
parser.add_argument("num", type=int, help='An integer that needs to get the square of')
parser.add_argument('--verbose',help='Print more info.', action='count')

args = parser.parse_args()

if args.verbose==2:
    print(f'The square of {args.num} is {args.num**2}')
elif args.verbose==1:
    print(f'{args.num}^2 == {args.num**2}')
else:
    print(args.num**2)
```

За это время мы можем определить, насколько подробным будет вывод, по количеству опций `--verbose` :

```
$ python test.py 9 --verbose --verbose
```

```
The square of 9 is 81
```

```
$ python test.py 9 --verbose 9^2 == 81
```

```
$ python test.py 9
```

```
81
```

Используйте "append" action

Действие `append` сохраняет список и добавляет в список каждое значение аргумента. В некоторых случаях будет полезно хранить несколько значений необязательного аргумента.

Чтобы использовать его, измените следующую строку предыдущего кода Python:

```
parser.add_argument('--verbose', help='Print more info.', action='append')
```

Теперь мы можем добавить несколько значений аргумента `--verbose` :

```
$ python test.py 9 --verbose 2 --verbose 3 --verbose 5
```

```
The verbose values are ['2', '3', '5']. The square of 9 is 81
```

6. Определяем ярлыки необязательных аргументов

Немного скучно много раз вводить `--verbose` . Как ярлык `-h` для опции `--help` . Мы также можем определить ярлык для нашей опции `--verbose` следующим образом:

```
parser.add_argument('-v', '--verbose', help='Print more info.', action='count')
```

После определения ярлыка `-v` мы можем использовать его напрямую:

```
$ python test.py 9 -vv
```

```
The square of 9 is 81
```

```
$ python test.py 9 -v
```

```
9^2 == 81
```

7. Работа с файлами в командной строке

Модуль `argparse` также может обрабатывать аргументы файлового типа, что делает удобным выполнение некоторых основных файловых операций через интерфейсы командной строки.

Чтение файла через командную строку

Если мы хотим отображать содержимое файла построчно на терминале, мы можем написать скрипт Python следующим образом:

```
import argparse

parser = argparse.ArgumentParser()
```

```
parser.add_argument('f', type=argparse.FileType('r'))
args = parser.parse_args()

for line in args.f:
    print(line.strip())
```

Теперь давайте выполним этот скрипт Python на терминале и передадим `test.txt` в качестве аргумента файлового типа:

```
$ python test.py test.txt
```

```
Yang is writing a post. Yang is writing on Medium.
```

Как показано выше, файл `test.txt` состоит из двух строк предложений. Они были правильно напечатаны на терминале с помощью нашего скрипта Python.

Запись данных в файл через командную строку

Чтобы записать данные в файл, нам нужно изменить оператор с `r` на `w`:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('f', type=argparse.FileType('w'))
args = parser.parse_args()

f = args.f
f.write('Yang is writing')
f.close()
```

Приведенный выше код запишет предложение «Ян пишет» в файл, назначенный через интерфейс командной строки. Использование его заключается в следующем:

```
$ python test.py test.txt
```

После выполнения вышеуказанной команды файл «`test.txt`» будет содержать одну строку предложения «Ян пишет».

Заключение

Освоение модуля `argparse` в Python поможет нам разрабатывать и реализовывать простые в использовании интерфейсы командной строки с помощью сценариев Python.

В тех случаях, когда нам нужен настраиваемый интерфейс командной строки, но мы не утруждаем себя написанием сложных сценариев `bash`, этот модуль Python — наш лучший друг.



Теги: `python`

Хабы: Настройка Linux, Python

**5**

Карма

2.4

Рейтинг

Алексей @alexxis

Системный администратор, программист

Подписаться



Хабр Фриланс

Комментарии 9

Публикации

ЛУЧШИЕ ЗА СУТКИ

ПОХОЖИЕ



Lunathecat

вчера в 12:00

Десятичный счетчик-дешифратор K561IE8 (CD4017) и красивый эффект на нём



Средний



8 мин



2.9K



+49



25



6



Holmogorov

20 часов назад

Инфернальная система. ОС Inferno, опередившая время

 Средний  7 мин  3.9K

 +38

 26

 3



SLY_G

20 часов назад

Какой инопланетные астрономы увидели бы Землю

 9 мин  4.3K

 +36

 19

 13



epeshk

18 часов назад

ArrayPool<T>: подводные камни

 12 мин  3.3K

 +34

 45

 2



Shyhartskoi

3 часа назад

Как разрабатывали StarCraft

 Простой  23 мин  3.5K

Ретроспектива

 +32

 16

 5

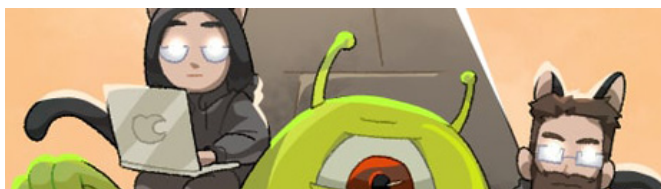
Разбираемся с проблемами в приложениях на дотнете. Инструкция про анализ с PerfView

Турбо

Показать еще


МИНУТОЧКУ ВНИМАНИЯ


Разместить








КУРСЫ

 Python-разработчик с нуля
4 апреля 2023 · 101 100 ₽ · Нетология

 Веб-разработчик с нуля
3 апреля 2023 · 161 100 ₽ · Нетология

 Java-разработчик с нуля
3 апреля 2023 · 109 500 ₽ · Нетология

 Иллюстрация и интерактивная графика
3 апреля 2023 · 111 900 ₽ · Нетология

 UX-дизайнер
3 апреля 2023 · 56 400 ₽ · Нетология

Больше курсов на Хабр Карьере

ЧИТАЮТ СЕЙЧАС

В строю ChatGPT клонов, которые можно крутить локально, прибыло. Встречайте gpt4all

 14K  16

Сотрудник «М.Видео» украл техники Apple на 2 млн рублей, чтобы расплатиться за скины в CS

 3.6K  21

Virgin Orbit прекращает деятельность и увольняет 85% сотрудников

 3.1K  4

Как разрабатывали StarCraft

 3.5K  5

Как подружить Алису с ChatGPT

 20K

 14


Разбираемся с проблемами в приложениях на дотнете. Инструкция про анализ с PerfView

Турбо

ИСТОРИИ



У вас спина белая: подборка смешных статей об IT



У вас спина белая


Хабр Карьера • Новости сервиса

Хабр Эксперты: IT-менторство на Хабр Карьере



Хабр Эксперты: IT-менторство на Хабр Карьере

Учёные из НАСА и двух университетов США объединились в поисках способа 3D-печати батарей из лунного реголита.



3D-печать батарей в космосе

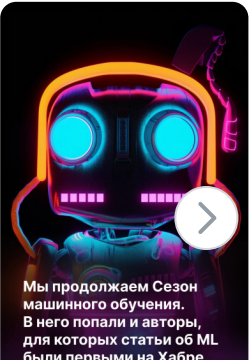


Клиент ChatGPT под MS-DOS


Его написал разработчик Йо Кхан Ман, известный фанат старых технологий. Он смог запустить чат-бота из командной строки на своем IBM 5155, выпущенном в 1984 году.



ChatGPT под MS-DOS



Мы продолжаем Сезон машинного обучения. В него попали и авторы, для которых статьи об ML были первыми на Хабре.



Сезон ML: Я здесь в первый раз

РАБОТА

Django разработчик
56 вакансий

Data Scientist
157 вакансий

Python разработчик
138 вакансий

Все вакансии

| | | | |
|-----------|------------|--------------------|--------------------|
| Профиль | Публикации | Устройство сайта | Корпоративный блог |
| Трекер | Новости | Для авторов | Медийная реклама |
| Диалоги | Хабы | Для компаний | Нативные проекты |
| Настройки | Компании | Документы | Образовательные |
| ППА | Авторы | Соглашение | программы |
| | Песочница | Конфиденциальность | Стартапам |
| | | | Мегапроекты |



Настройка языка

Техническая поддержка

Вернуться на старую версию