Сообщить об ошибке.

хочу помочь проекту **[**

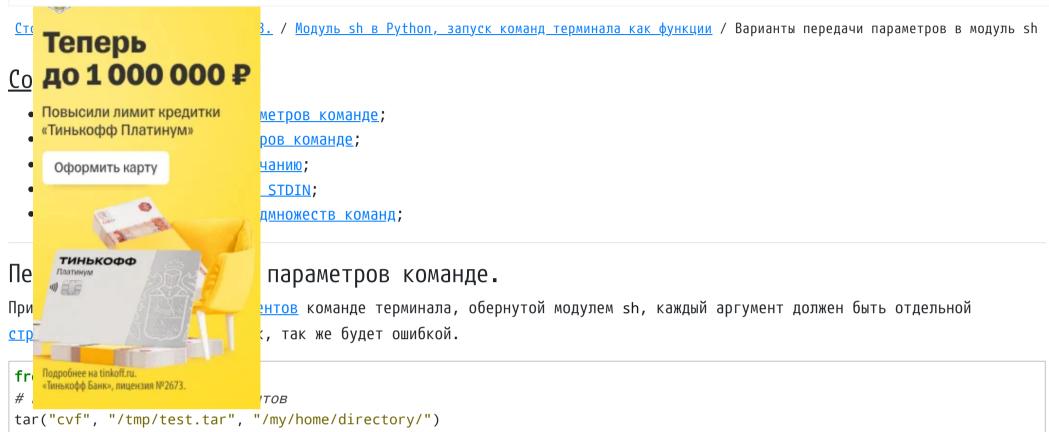
вкту Варианты передачи параметров в модуль sh



3D карта мира из дерева по специальной цене!

Отличный подарок вам и вашим близким. Заказывайте на сайте используя промокод!

Купить



При передаче аргументов, как указано ниже, будет возникать ошибка:

```
from sh import tar
# неправильная передача aprymeнтов
tar("cvf /tmp/test.tar /my/home/directory")
# Traceback (most recent call last):
# File "<stdin>", line 1, in <module>
# ...
# raise exc
# sh.ErrorReturnCode_2:
# RAN: /bin/tar 'cvf /tmp/test.tar /my/home/directory'
```

Почему параметры должны быть отдельными строками?

Это сбивает с толку многих новых пользователей модуля sh. Они хотят сделать что-то подобное tar("cvf /tmp/test.tar /my/home/directory") и ожидают, что это просто сработает, Но вместо этого они получат сбивающее с толку сообщение об ошибке (как показано выше).

Причина, по которой они ожидают, что это сработает, заключается в том, что оболочки, такие как Bash, автоматически анализируют gthtlfyye. командную строку и разбивают ее на аргументы, прежде чем отправлять их в исполняемый файл. У них есть сложный набор правил, некоторые из которых представлены Python модулем shlex.

Даже если бы разработчики модуля хотели реализовать это в sh, то это уменьшило бы возможности пользователей параметризовать части аргументов. Им пришлось бы использовать строковую интерполяцию, которая по сути уродлива и подвержена ошибкам:

```
from sh import tar
tar("cvf {1} {2}".format("/tmp/tar1.tar", "/home/oh no a space")
```

В приведенном выше примере "/home/oh", "no", "a" и "space" будут отдельными аргументами для команды терминала tar, что приведет к неожиданному поведению программы. В принципе, каждая команда с позиционными параметрами должна была бы ожидать символов, которые могли бы сломать синтаксический анализатор.

Передача параметров команде по ключу.

Мо. Вверх поддерживает ключи короткой формы (например -a) и длинной формы (например --arg) в качестве параметров командной строки:

```
# вызывает "curl http://duckduckgo.com/ -o page.html --silent"
curl("http://duckduckgo.com/", o="page.html", silent=True)

# или если не использовать ключевые параметры
curl("http://duckduckgo.com/", "-o", "page.html", "--silent")

# вызывает "adduser amoffat --system --shell=/bin/bash --no-create-home"
adduser("amoffat", system=True, shell="/bin/bash", no_create_home=True)

# без использования ключевых параметров
adduser("amoffat", "--system", "--shell", "/bin/bash", "--no-create-home")
```

Как упорядочивать ключевые параметры командной строки?

Обычно этот вопрос задают, когда пользователь пытается выполнить что-то вроде следующей командной строки:

```
my-command --arg1=val1 arg2 --arg3=val3
```

Первая попытка, которую пользователь пытается сделать это :

```
import sh
sh.bash_command(arg1="val1", "arg2", arg3="val3")
```

Это не работает, потому что в Python позиционные аргументы, такие как arg2, не могут идти после ключевых аргументов.

Кроме того, вполне возможно, что --arg3=val3 предшествует --arg1=val1. Причина этого заключается в том, что **kwargs в функции является неупорядоченным отображением (словарем), и поэтому пары ключ-значение не гарантированно разрешаются в определенном порядке.

Таким образом, в данном случае, решение состоит в том, чтобы отказаться от использования ключевых аргументов и просто использовать необработанные позиционные аргументы:

```
sh.bash_command("--arg1=val1", "arg2", "--arg3=val3")
```

Параметры команды по умолчанию.

Часто встает необходимость переопределить параметры по умолчанию для всех команд терминала, запускаемых через модуль sh. Например, предположим, что надо объединять вывод всех команд в буфер <u>io.StringIO</u>.

```
import sh
from io import StringIO

buf = StringIO()

sh.ls("/", _out=buf)
sh.whoami(_out=buf)
sh.ps("auxwf", _out=buf)
```

Из кода видно, что постоянная передача _out=buf во все вызовы быстро надоедает. К счастью, можно создавать контексты выполнения, которые позволяют устанавливать аргументы по умолчанию для всех команд, порожденных из этого контекста:

```
import sh
from io import StringIO

buf = StringIO()
sh_buf = sh(_out=buf)

sh_buf.ls("/")
sh_buf.whoami()
sh_buf.ps("auxwf")
```

Теперь все, что запускается из sh_buf, будет посылать свои выходные данные в экземпляр буфера StringIO.

Контексты выполнения также могут быть импортированы, например, из модуля нижнего уровня buf.py:

```
# buf.py
import sh
free import StringIO
buf = StringIO()
```

```
12.09.2023, 13:46
```

```
sh_buf = sh(_out=buf)

# cmd.py
from sh_buf import ls, whoami, ps

ls("/")
whoami()
ps("auxwf")
```

Передача входных данных команде через STDIN

Поток STDIN отправляется процессу непосредственно с помощью специального ключевого аргументы _in:

```
print(cat(_in="test"))
# test
```

Любая команда терминала, принимающая входные данные от STDIN, может быть использована следующим образом:

```
print(tr("[:lower:]", "[:upper:]", _in="sh is awesome"))
# SHISAWESOME
```

Следовательно пользователи не ограничены использованием только <u>строк</u> в качестве входных данных (но не параметров). Можно использовать файловый объект, очередь <u>Queue</u> или любую <u>итерацию</u> (<u>список</u>, <u>словарь</u> и т. д.):

```
stdin = ["sh", "is", "awesome"]
out = tr("[:lower:]", "[:upper:]", _in=stdin)
```

Выполнение собственных подмножеств команд.

Многие программы терминала имеют свои собственные подмножества команд, такие как git (например branch или checkout), svn (например update или status) и sudo (где любая команда, следующая за sudo, считается ее подкомандой). Модуль sh обрабатывает подкоманды через синтаксис Python доступа к атрибутам:

```
from sh import git, sudo

# выполнит "git branch -v"
print(git.branch("-v"))
# та же команда
print(git("branch", "-v"))

# выполнит "sudo /bin/ls /root"
print(sudo.ls("/root"))
# та же команда
print(sudo("/bin/ls", "/root"))
```

Обращение к подкомандам через синтаксис доступа к атрибутам - это в основном синтаксический сахар, который делает вызов некоторых программ концептуально более приятным.

Примечание. Если будете использовать sudo в качестве подкоманды, обязательно посмотрите раздел "<u>Использование sudo в</u> модуле sh".

Содержание раздела:

- ОБЗОРНАЯ СТРАНИЦА РАЗДЕЛА
- <u>Передача параметров в команды модуля sh</u>
- <u>Коды завершения и исключения модуля sh</u>
- <u>Перенаправление вывода команды модуля sh</u>
- <u>Фоновое выполнение команды с модулем sh</u>
- <u>Конвейер в стиле Bash и модуль sh</u>
- <u>Использование sudo с модулем sh</u>
- Использование ssh с модулем sh
- <u>Специальные ключевые аргументы модуля sh</u>
- <u>Объект запущенной команды модуля sh</u>
- Bверх | mmand() модуля sh

• <u>Частые вопросы по модулю sh</u>

DOCS-Python.ru™, 2023 г.

(Внимание! При копировании материала ссылка на источник обязательна)

@docs_python_ru

Вверх