



```
# 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 [io.StringIO](#).

```
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
from io import StringIO

buf = StringIO()
```

Вверх

```
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 [строк](#) в качестве входных данных (но не параметров). Можно использовать [файловый объект](#), очередь [Queue](#) или любую [итерацию](#) ([список](#), [словарь](#) 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 ["Использование sudo в модуле sh"](#).

### Содержание раздела:

- [ОБЗОРНАЯ СТРАНИЦА РАЗДЕЛА](#)
- [Передача параметров в команды модуля sh](#)
- [Коды завершения и исключения модуля sh](#)
- [Перенаправление вывода команды модуля sh](#)
- [Фоновое выполнение команды с модулем sh](#)
- [Конвейер в стиле Bash и модуль sh](#)
- [Использование sudo с модулем sh](#)
- [Использование ssh с модулем sh](#)
- [Специальные ключевые аргументы модуля sh](#)
- [Объект запущенной команды модуля sh](#)
- [Метод Command\(\) модуля sh](#)

Вверх

- [Частые вопросы по модулю sh](#)

Вверх