

Работа с командной строкой (скрипты, аргументы)

План урока

- 1 Про командную строку
- 2 Виртуальные машины
- 3 Запуск командной строки
- 4 Принципы работы с командной строкой
- 5 Базовые команды
- 6 Потоки, exit-коды
- 7 Как запустить свой скрипт?
- 8 Простой парсинг аргументов в Python

Аннотация

В уроке рассказывается о командной строке, как самом простом, с точки зрения реализации, интерфейсе пользователя, 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**.

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.

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

Какие же преимущества имеет командная строка перед GUI?

1. Простота удалённой работы. Легче синхронизировать два экрана разного разрешения, которые выводят текст, чем два экрана, что выводят 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 неё 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.

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

2. Виртуальные машины

Очень часто у программиста или IT-инженера возникает задача смоделировать работу другой операционной системы по отношению к операционной системе, установленной на своём компьютере. 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 как бы *операционную систему 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 подготовили 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.

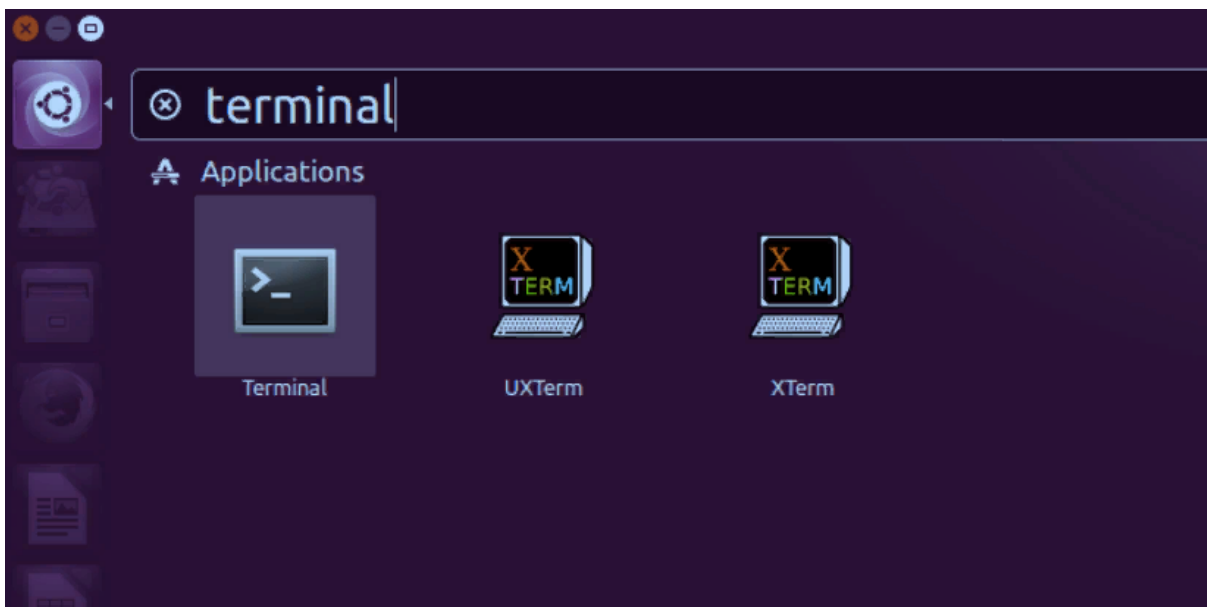
По [ссылке](#) вы можете скачать VirtualBox для MacOS и Windows, а также готовые образы ОС Ubuntu 17.04 и ОС Lite Linux для VirtualBox. (Это — zip-архив, и он немаленький).

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

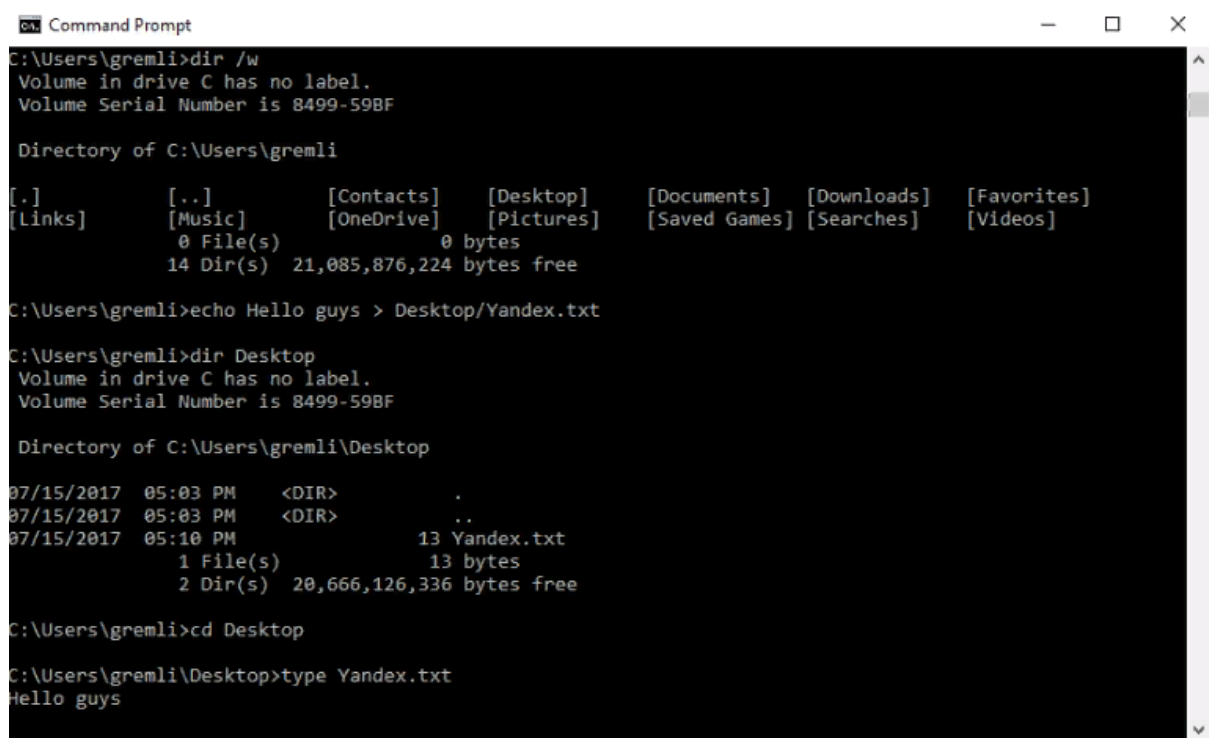
А в этом [видео](#) показывается, как подключить готовый образ к VirtualBox.

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 её ни разу не видели, можете быть уверены, что и в вашей системе она есть. 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):



```
CA Command Prompt
C:\Users\greml>dir /w
Volume in drive C has no label.
Volume Serial Number is 8499-59BF

Directory of C:\Users\greml

[.]          [..]          [Contacts]    [Desktop]     [Documents]  [Downloads]  [Favorites]
[Links]      [Music]       [OneDrive]    [Pictures]    [Saved Games] [Searches]   [Videos]
             0 File(s)      0 bytes
             14 Dir(s)  21,085,876,224 bytes free

C:\Users\greml>echo Hello guys > Desktop\Yandex.txt

C:\Users\greml>dir Desktop
Volume in drive C has no label.
Volume Serial Number is 8499-59BF

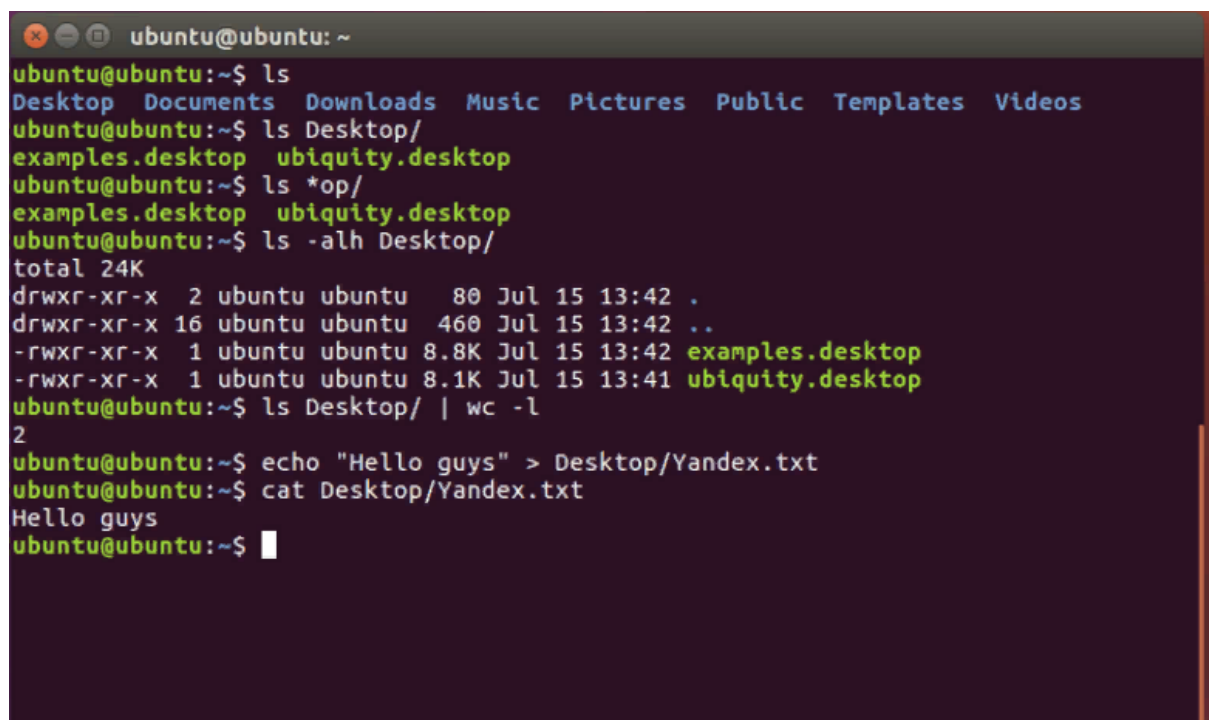
Directory of C:\Users\greml\Desktop

07/15/2017  05:03 PM  <DIR>          .
07/15/2017  05:03 PM  <DIR>          ..
07/15/2017  05:10 PM                13 Yandex.txt
             1 File(s)      13 bytes
             2 Dir(s)  20,666,126,336 bytes free

C:\Users\greml>cd Desktop

C:\Users\greml\Desktop>type Yandex.txt
Hello guys
```

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



```
ubuntu@ubuntu: ~
ubuntu@ubuntu:~$ ls
Desktop Documents Downloads Music Pictures Public Templates Videos
ubuntu@ubuntu:~$ ls Desktop/
examples.desktop ubiquity.desktop
ubuntu@ubuntu:~$ ls *op/
examples.desktop ubiquity.desktop
ubuntu@ubuntu:~$ ls -alh Desktop/
total 24K
drwxr-xr-x  2 ubuntu ubuntu   80 Jul 15 13:42 .
drwxr-xr-x 16 ubuntu ubuntu  460 Jul 15 13:42 ..
-rwxr-xr-x  1 ubuntu ubuntu 8.8K Jul 15 13:42 examples.desktop
-rwxr-xr-x  1 ubuntu ubuntu 8.1K Jul 15 13:41 ubiquity.desktop
ubuntu@ubuntu:~$ ls Desktop/ | wc -l
2
ubuntu@ubuntu:~$ echo "Hello guys" > Desktop/Yandex.txt
ubuntu@ubuntu:~$ cat Desktop/Yandex.txt
Hello guys
ubuntu@ubuntu:~$
```

4. Принципы работы с командной строкой

После запуска программы командной строки (её ещё называют **терминалом**, **консолью** или **shell**) на экране появляется чёрное окно (обычно, оно чёрное) с приглашением ввода. Это командная оболочка, которая будет выполнять наши команды. В данный момент операционная система ждёт 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**.

Давайте рассмотрим команду вывода содержимого директории **ls -alh Desktop/**. Результат её 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 ещё 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 -l 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 её имя. В большинстве случаев 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 — *переменная окружения*.

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

С помощью **whereis** можно узнать, где в файловой системе располагается нужная нам команда.

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

```
ubuntu@ubuntu:~$ echo $PATH
/home/ubuntu/bin:/home/ubuntu/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
ubuntu@ubuntu:~$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
ubuntu@ubuntu:~$ /bin/ls Desktop/
examples.desktop  ubiquity.desktop
ubuntu@ubuntu:~$
```

5. Базовые команды

Приведём список самых важных и часто используемых команд, которые стоит запомнить (в скобках 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 произойдёт 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;
- `..` — 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 вами с ним встречались) и выводить данные на **стандартный поток вывода** — `stdout`.

Но, кроме этих двух потоков, существует ещё 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, в принципе) выполнена успешно. Если же значение Exitcode больше нуля, то 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 $?`.

7. Как запустить свой скрипт?

Когда вы написали свою программу на языке Python и хотите её 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 ещё не всё. 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)

~/ЯндексЛицей/materials/additional//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.

На этом мы пока остановимся и перейдем к решению задач.

[Помощь](#)

© 2018 – 2019 ООО «Яндекс»