

Лекция 4. Работа в оболочке ОС, написание скриптов

Коновалов А.В.

9 апреля 2023 г.

Работа в текстовой оболочке операционной системы (ОС)

Оболочки операционной системы (ОС) (1)

Оболочка операционной системы — программа, посредством которой пользователь может осуществлять взаимодействие с ОС: запускать другие программы и работать с файлами.

В современных ОС используются графические оболочки, однако есть возможность и текстовые.

Графические оболочки, как правило, не предоставляют механизмов для автоматизации повторяющихся действий, все действия выполняются интерактивно, при участии пользователя.

Текстовые оболочки допускают как интерактивный режим — пользователь вводит команды и наблюдает их результат, так и «пакетный» (batch) — пользователь может создать файл со списком команд, которые будут выполняться автоматически. Кроме того, в текстовых оболочках присутствуют средства, позволяющие комбинировать работу нескольких программ — вывод одной программы связывать со входом другой.

Оболочки операционной системы (ОС) (2)

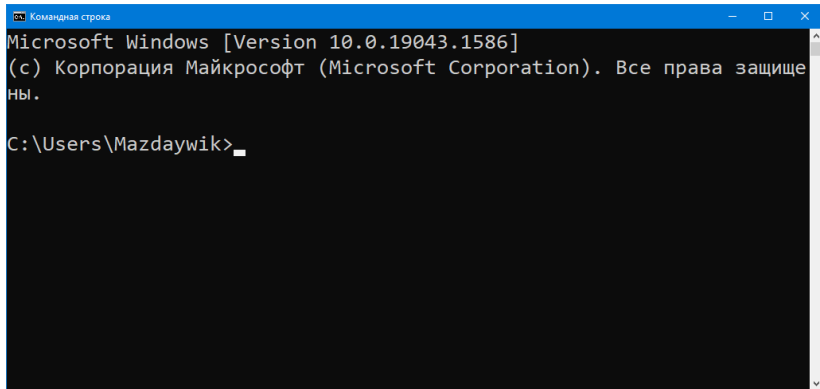
На сегодняшней лекции мы будем рассматривать основные команды и принципы работы в оболочках `cmd.exe` на Windows и `Bash` на Linux и macOS.

Для запуска оболочки `cmd.exe` на Windows нужно найти пункт меню «Командная строка» в группе «Служебные — Windows» в меню «Пуск»:

- ▶ «Пуск» → «Служебные — Windows» → «Командная строка» (Windows 10)
- ▶ «Пуск» → «Программы» → «Служебные» → «Командная строка (Windows 7)»

Для запуска оболочки на Linux и macOS нужно найти приложение «Терминал» или «Консоль» (в разных дистрибутивах Линукса оно может называться по-разному)

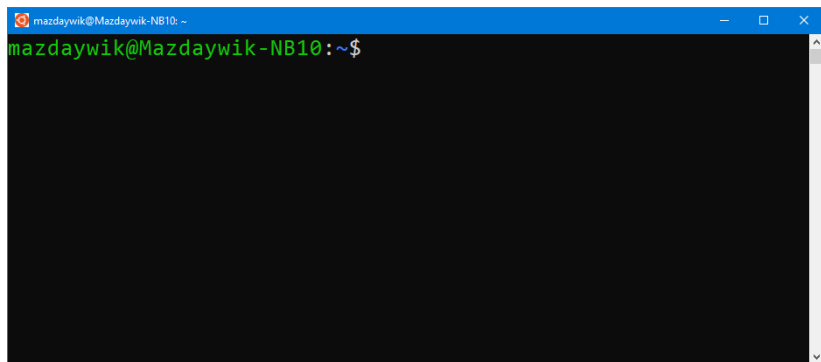
Командная строка Windows

A screenshot of the Windows Command Prompt window. The title bar is blue and contains the text "Командная строка" (Command Prompt) on the left and standard window control buttons (minimize, maximize, close) on the right. The main area has a black background with white text. The text displayed is: "Microsoft Windows [Version 10.0.19043.1586]" followed by "(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены." (All rights reserved.) on the next line. Below this, the current directory is shown as "C:\Users\Mazdaywik>" with a white cursor at the end.

```
Microsoft Windows [Version 10.0.19043.1586]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

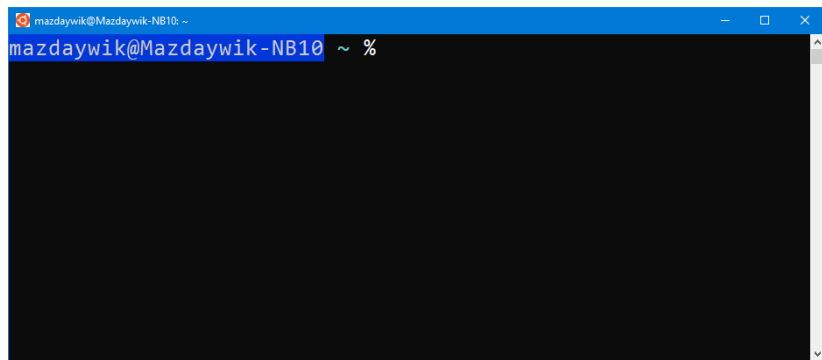
C:\Users\Mazdaywik>
```

Оболочка Bash — командная строка Linux или macOS



Bash по умолчанию используется на большинстве дистрибутивов Linux и старых версиях macOS.

Оболочка Zsh — командная строка Linux или macOS



Zsh по умолчанию используется на современных версиях macOS.

Базовые команды в Bash и Zsh совпадают, поэтому на лекции мы будем рассматривать только команды Bash.

Приглашение оболочки ОС (1)

У оболочки есть так называемое **приглашение** — место на экране для ввода команды. В командной строке Windows приглашение выглядит так:

```
C:\Users\Mazdaywik>
```

В нём отображается путь к текущей папке (у оболочки, как и у любой другой программы, есть текущая папка) и знак >. Пользователь вводит команды после знака >.

Приглашение оболочки ОС (2)

В оболочке Bash (используемой по умолчанию на Linux и старых версиях macOS), приглашение выглядит так:

```
mazdaywik@Mazdaywik-NB10:~$
```

В нём отображается имя текущего пользователя, имя компьютера, знак :, путь к текущей папке (знаком ~ отображается домашняя папка) и знак \$, после которого пользователь вводит команды.

В современных версиях macOS используется оболочка Zsh, её приглашение похоже на приглашение Bash, но завершается на знак %:

```
mazdaywik@Mazdaywik-NB10 ~ %
```

Команды в оболочке ОС

Команды оболочки позволяют перемещаться по папкам, манипулировать файлами (копировать, переименовывать, перемещать, удалять) и запускать другие программы.

Общий синтаксис команды:

⟨команда⟩ ⟨параметр1⟩ ⟨параметр2⟩...

Здесь ⟨команда⟩ — это имя команды, параметры — указание режимов работы команды и имён файлов, которыми команда манипулирует. Команды могут быть без параметров.

Имя команды отделяется от параметров пробелом, сами параметры тоже разделяются пробелами. Если имя команды или параметр сам должен содержать пробелы, то он заключается в кавычки: "двойные" (работают и на Windows, и на Linux/macOS) или 'одинарные' (работают только на Linux/macOS).

Команды бывают встроенные — обрабатываются самой оболочкой, либо являются именами программ.

Основные команды оболочки

- ▶ Смена текущей папки, `cd`
- ▶ Создание папки, `mkdir`
- ▶ Создание текстового файла
- ▶ Просмотр содержимого папки
- ▶ Копирование файлов
- ▶ Переименование файлов
- ▶ Перемещение файлов
- ▶ Удаление файлов и папок
- ▶ Просмотр содержимого файла
- ▶ Очистка экрана
- ▶ Завершение работы в оболочке

Смена текущей папки, cd

`cd <имя папки>`, работает и на Windows, и на Linux, и на macOS.

```
C:\Users\Mazdaywik>cd Desktop
```

```
C:\Users\Mazdaywik\Desktop>
```

```
mazdaywik@Mazdaywik-NB10:~$ cd example
```

```
mazdaywik@Mazdaywik-NB10:~/example$
```

По приглашению командной строки видно, что текущая папка изменилась. В качестве имени целевой папки можно использовать `..` для перехода в родительскую папку.

Создание папки, mkdir

Для того, чтобы создать новую пустую папку, используется команда `mkdir`:

```
mkdir <имя новой папки>
```

На Windows вместо `mkdir` можно использовать более короткий синоним `md`, на Linux и macOS — только `mkdir`.

```
mazdaywik@Mazdaywik-NB10:~$ mkdir "New folder"
```

```
mazdaywik@Mazdaywik-NB10:~$ cd "New folder"
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$
```

```
C:\Users\Mazdaywik>mkdir "New folder"
```

```
C:\Users\Mazdaywik>cd "New folder"
```

```
C:\Users\Mazdaywik\New folder>
```

Создание текстового файла (1)

На Windows для создания текстового файла используется команда вида

```
copy con <имя файла>
```

после чего с клавиатуры нужно ввести содержимое файла. Ввод завершается нажатием клавиш Ctrl-Z и Enter.

```
C:\Users\Mazdaywik\New folder>copy con "new file.txt"  
first line  
second line  
third line  
^Z
```

```
Скопировано файлов:           1.
```

```
C:\Users\Mazdaywik\New folder>
```

Создание текстового файла (2)

На Linux и macOS нужно ввести команду

```
cat > <имя файла>
```

ввести содержимое файла и нажать комбинацию клавиш Ctrl-D.

```
mazdaywik@Mazdaywik-NB10:~/New folder$ cat > "new file.txt"
first line
second line
third line
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Просмотр содержимого папки (1)

На Windows используется команда `dir`, которая распечатывает содержимое папки, выводя сведения о каждом файле.

Если команду вызвать с параметром `/b`, будут выведены только имена файлов. В папке `New folder` на данный момент всего один файл:

```
C:\Users\Mazdaywik\New folder>dir
```

Том в устройстве C имеет метку System

Серийный номер тома: 0052-DDD1

Содержимое папки C:\Users\Mazdaywik\New folder

09.04.2022	14:33	<DIR>	.
09.04.2022	14:33	<DIR>	..
09.04.2022	14:33		37 new file.txt
		1 файлов	37 байт
		2 папок	43 388 399 616 байт свободно

Просмотр содержимого папки (2)

Если команду вызвать с параметром /b, будут выведены только имена файлов. В папке New folder на данный момент всего один файл:

```
C:\Users\Mazdaywik\New folder>dir /b  
new file.txt
```

```
C:\Users\Mazdaywik\New folder>
```

Просмотр содержимого папки (3)

На Linux и macOS используется команда `ls`, которая распечатывает имена файлов в папке. Если указать параметр `-l`, то будут распечатаны подробные сведения:

```
mazdaywik@Mazdaywik-NB10:~/New folder$ ls
```

```
'new file.txt'
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ ls -l
```

```
total 4
```

```
-rw-r--r-- 1 mazdaywik mazdaywik 34 Apr  9 14:34 'new file.txt'
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Копирование файлов (1)

На Windows используется команда

copy <исходный файл> <имя копии>

copy <исходный файл> <целевая папка>

На Linux и macOS:

cp <исходный файл> <имя копии>

cp <исходный файл> <целевая папка>

Копирование файлов (2)

Для примера скопируем файл new file.txt в copy.txt:

```
C:\Users\Mazdaywik\New folder>copy "new file.txt" copy.txt
```

Скопировано файлов: 1.

```
C:\Users\Mazdaywik\New folder>dir
```

Том в устройстве C имеет метку System

Серийный номер тома: 0052-DDD1

Содержимое папки C:\Users\Mazdaywik\New folder

09.04.2022	14:41	<DIR>	.
09.04.2022	14:41	<DIR>	..
09.04.2022	14:33		37 copy.txt
09.04.2022	14:33		37 new file.txt
		2 файлов	74 байт
		2 папок	43 390 611 456 байт свободно

Копирование файлов (3)

Команда `dir` нам показала, что в папке теперь находятся два файла. Имя файла `new file.txt` нужно было заключать в кавычки, т.к. оно содержит пробел. Имя целевого файла `copy.txt` пробелов не содержит, поэтому кавычки не нужны.

На Linux:

```
mazdaywik@Mazdaywik-NB10:~/New folder$ cp "new file.txt" copy
mazdaywik@Mazdaywik-NB10:~/New folder$ ls -l
total 8
-rw-r--r-- 1 mazdaywik mazdaywik 34 Apr  9 14:43  copy.txt
-rw-r--r-- 1 mazdaywik mazdaywik 34 Apr  9 14:34  'new file.tx
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Переименование файлов (1)

На Windows используется команда

```
ren <исходное имя> <новое имя>
```

На Linux и macOS:

```
mv <исходное имя> <новое имя>
```

Переименование файлов (2)

Переименуем файл copy.txt в backup.txt:

```
C:\Users\Mazdaywik\New folder>ren copy.txt backup.txt
```

```
C:\Users\Mazdaywik\New folder>dir /b
```

```
backup.txt
```

```
new file.txt
```

```
C:\Users\Mazdaywik\New folder>
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ mv copy.txt backup.txt
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ ls
```

```
backup.txt  'new file.txt'
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Перемещение файлов (1)

На Windows используется команда move:

```
move <исходный файл> <целевой файл>
```

```
move <исходный файл> <целевая папка>
```

на Linux и macOS — mv:

```
mv <исходный файл> <целевой файл>
```

```
mv <исходный файл> <целевая папка>
```

Таким образом, на Linux и macOS используется одна и та же команда и для переименования, и для перемещения файлов.

Перемещение файлов (2)

Создадим папку nested в текущей папке и переместим в неё файл backup.txt:

```
C:\Users\Mazdaywik\New folder>mkdir nested
```

```
C:\Users\Mazdaywik\New folder>move backup.txt nested
```

```
Перемещено файлов:          1.
```

```
C:\Users\Mazdaywik\New folder>dir /b
```

```
nested
```

```
new file.txt
```

```
C:\Users\Mazdaywik\New folder>dir /b nested
```

```
backup.txt
```

```
C:\Users\Mazdaywik\New folder>
```

Перемещение файлов (3)

Мы создали папку `nested`, переместили в неё файл `backup.txt`, посмотрели содержимое текущей папки и содержимое папки `nested`. Как видим, команда `dir` может принимать имя папки, содержимое которой нужно распечатать. Если имя не указано, то распечатывается содержимое текущей папки.

Перемещение файлов (4)

```
mazdaywik@Mazdaywik-NB10:~/New folder$ mkdir nested
mazdaywik@Mazdaywik-NB10:~/New folder$ mv backup.txt nested
mazdaywik@Mazdaywik-NB10:~/New folder$ ls
nested  'new file.txt'
mazdaywik@Mazdaywik-NB10:~/New folder$ ls nested
backup.txt
mazdaywik@Mazdaywik-NB10:~/New folder$
```

На Linux и macOS команда `ls` также поддерживает указание имени папки, которую нужно распечатать.

Удаление файлов (1)

На Windows используется команда `del` или `erase` (это два синонима), которая принимает имя удаляемого файла:

```
del <имя файла>
```

```
erase <имя файла>
```

На Linux и macOS такая команда называется `rm`:

```
rm <имя файла>
```

Удаление файлов (2)

Удалим файл backup.txt в папке nested:

```
C:\Users\Mazdaywik\New folder>del nested\backup.txt
```

```
C:\Users\Mazdaywik\New folder>dir /b nested
```

```
C:\Users\Mazdaywik\New folder>
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ rm nested/backup.txt
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ ls nested
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Команды dir /b и ls ничего не вывели, т.к. папки пустые.

Удаление папок (1)

Для удаления папки используется команда `rmdir` на всех трёх рассматриваемых операционных системах. На Windows можно использовать более короткий синоним `rd`. Команда `rmdir (rd)` может удалить только пустой каталог, в противном случае выдаст ошибку.

Удаление папок (2)

Удалим папку nested:

```
C:\Users\Mazdaywik\New folder>rmdir nested
```

```
C:\Users\Mazdaywik\New folder>dir /b  
new file.txt
```

```
C:\Users\Mazdaywik\New folder>
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ rmdir nested
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ ls
```

```
'new file.txt'
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Просмотр содержимого файла (1)

Для того, чтобы вывести на экран содержимое файла, на Windows используется команда type:

```
type <имя файла>
```

На Linux и macOS — cat:

```
cat <имя файла>
```


Просмотр содержимого файла (2)

Посмотрим содержимое файла new file.txt:

```
C:\Users\Mazdaywik\New folder>type "new file.txt"
```

```
first line
```

```
second line
```

```
third line
```

```
C:\Users\Mazdaywik\New folder>
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$ cat "new file.txt"
```

```
first line
```

```
second line
```

```
third line
```

```
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Очистка экрана и завершение работы в оболочке

Чтобы стереть содержимое окна консоли, нужно выполнить команду `cls` на Windows или `clear` на Linux/macOS.

Команда `exit` (работает везде) закрывает окно оболочки.

Запуск консоли Python в консоли оболочки (1)

На Windows, если Python установлен правильно, нужно ввести команду `python`, на Linux и macOS — `python3` (т.к. команда `python` без цифры на конце может запустить Python 2.7 на некоторых дистрибутивах Linux или версиях macOS).

```
C:\Users\Mazdaywik\New folder>python
Python 3.9.5 (tags/v3.9.5:0a7dcbb, May 3 2021, 17:27:52) [MSC v
Type "help", "copyright", "credits" or "license" for more inform
>>> 1+2
3
>>> exit
Use exit() or Ctrl-Z plus Return to exit
>>> exit()
```

```
C:\Users\Mazdaywik\New folder>
```

Запуск консоли Python в консоли оболочки (2)

```
mazdaywik@Mazdaywik-NB10:~/New folder$ python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information
>>> 1+2
3
>>> exit
Use exit() or Ctrl-D (i.e. EOF) to exit
>>> exit()
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Запуск программ на Python в оболочке (1)

Для запуска программы на Python'е нужно выполнить в оболочке команду

```
python <имя исходного файла>
```

```
python3 <имя исходного файла>
```

в зависимости от операционной системы.

Запуск программ на Python в оболочке (2)

Напишем программу, которая распечатывает фразу Hello, World! и запустим её.

```
C:\Users\Mazdaywik\New folder>copy con hello.py  
print('Hello, World!')  
^Z
```

Скопировано файлов: 1.

```
C:\Users\Mazdaywik\New folder>python hello.py  
Hello, World!
```

```
C:\Users\Mazdaywik\New folder>
```

Запуск программ на Python в оболочке (3)

```
mazdaywik@Mazdaywik-NB10:~/New folder$ cat > hello.py  
print('Hello, World!')  
mazdaywik@Mazdaywik-NB10:~/New folder$ python3 hello.py  
Hello, World!  
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Запуск программ на Python в оболочке (4)

На Windows программы на Python можно запускать, просто указывая имя исходного файла — интерпретатор Python будет вызываться автоматически:

```
C:\Users\Mazdaywik\New folder>hello.py  
Hello, World!
```

```
C:\Users\Mazdaywik\New folder>
```


Запуск программ на Python в оболочке (5)

На Linux и macOS, чтобы так сделать, нужно выполнить дополнительные действия: добавить в начало файла специальный комментарий

```
#!/usr/bin/env python3
```

и пометить файл как исполнимый командой `chmod +x`.

```
mazdaywik@Mazdaywik-NB10:~/New folder$ cat > hello.py  
#!/usr/bin/env python3
```

```
print('Hello, World!')
```

Запуск программ на Python в оболочке (6)

```
mazdaywik@Mazdaywik-NB10:~/New folder$ chmod +x hello.py
mazdaywik@Mazdaywik-NB10:~/New folder$ ./hello.py
Hello, World!
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Одна из особенностей оболочек Linux и macOS — для запуска программ из текущей папки нужно явно добавлять в начало `./`, к этому надо привыкнуть.

Взаимодействие между оболочкой и Python

Стандартные потоки ввода-вывода (1)

В операционных системах (и в Windows, и в unix-подобных системах macOS и Linux) для программ, запускаемых в консоли, доступны три стандартных потока ввода-вывода — три псевдофайла, посредством которых программы могут взаимодействовать с пользователем и друг с другом.

- ▶ Стандартный ввод (условно называемый `stdin`) — по умолчанию связан с вводом с клавиатуры.
- ▶ Стандартный вывод (`stdout`) — по умолчанию связан с выводом текста на экран, предназначен для вывода полезных данных программы.
- ▶ Стандартный вывод ошибок (`stderr`) — тоже по умолчанию связан с экраном, предназначен для вывода сообщений об ошибках.

Если программа запрашивает пользовательский ввод, то, как правило, она читает его из (псевдо)файла `stdin`, если печатает на экран — по умолчанию выводит в (псевдо)файл `stdout`.

Стандартные потоки ввода-вывода (2)

С точки зрения программы, стандартные потоки являются разновидностью файлов, в большинстве языков программирования с ними можно работать точно также, как и с обычными файлами. И у стандартного ввода тоже есть возможность достигнуть конца файла.

Для того, чтобы ввести конец файла в операционной системе Windows, нужно нажать комбинацию клавиш Ctrl-Z (на экране высветится ^Z) и после этого Enter. Для того, чтобы ввести конец файла в unix-подобных ОС, нужно нажать Ctrl-D.

Стандартные потоки ввода-вывода (3) — стандартный ввод

Командная строка позволяет связывать стандартные потоки с файлами и даже между собой.

Пусть есть некоторая команда `program`, которая принимает некоторые аргументы, кроме того, в процессе работы запрашивает пользовательский ввод с клавиатуры и выводит что-то на экран.

```
program arg1 arg2 ...
```

Для того, чтобы подменить пользовательский ввод на чтение текстового файла, нужно к команде запуска программы добавить `< «имя-файла.txt»:`

```
program arg1 arg2 ... < input.txt
```

Стандартные потоки ввода-вывода (4) — стандартный ввод

При таком запуске программа не будет запрашивать у пользователя ничего с клавиатуры, вместо этого операции чтения будут читать строки из файла.

Можно считать, что при перенаправлении ввода стандартный ввод неявно считается файлом, открытым в режиме `r` (для чтения).

Рассмотрим следующую программу `greeting.py`:

```
name = input("Введите ваше имя:")  
sirname = input("Введите вашу фамилию:")  
  
print("Привет,", name, sirname)
```

(встроенная функция `input()` запрашивает ввод с клавиатуры)

Стандартные потоки ввода-вывода (5) — стандартный ввод

Рассмотрим запуск программы обычным образом и с использованием перенаправления ввода:

```
Z:\>python greeting.py
```

```
Введите ваше имя:Александр
```

```
Введите вашу фамилию:Коновалов
```

```
Привет, Александр Коновалов
```

```
Z:\>type input.txt
```

```
Alexander
```

```
Konovarov
```

```
Z:\>python greeting.py < input.txt
```

```
Введите ваше имя:Введите вашу фамилию:Привет, Alexander Konov
```

```
Z:\>
```


Стандартные потоки ввода-вывода (4) — стандартный ввод

Первый запуск демонстрирует ввод данных с клавиатуры (`stdin` связан с клавиатурой): я вводил свои имя и фамилию (завершая ввод каждой строки нажатием на клавишу Enter, которая вводила и перевод строки тоже), набираемый мною текст отображался на экране и считывался из `stdin` встроенной функцией `input()`.

Далее, при помощи команды `type` я показал, что в данной папке есть текстовый файл `input.txt` с двумя строчками, в которых написаны, соответственно, имя и фамилия (из-за особенностей Windows, кириллица может считываться неверно).

В третьей команде я связал стандартный ввод `stdin` с файлом `input.txt`, в результате чего программа считывала данные не с клавиатурного ввода, а из текстового файла — на экране видно только то, что она выводит, но не считывает.

Стандартные потоки ввода-вывода (5) — стандартный вывод

Стандартный вывод можно перенаправить двумя способами: для перезаписи и для дозаписи (сравни с режимами `w` и `a` функции `open()` в Python). Для того, чтобы стандартный вывод перенаправить в файл в режиме перезаписи (если файла не было, он будет создан, если файл был, его старое содержимое сотрётся), нужно к команде запуска добавить `>` «имя-файла.txt», в режиме дозаписи (если файл был, то новые данные будут добавляться в конец) — `>>` «имя-файла.txt»:

```
program arg1 arg2 ... > output.txt
```

```
program arg1 arg2 ... >> output.txt
```

Стандартные потоки ввода-вывода (5) — стандартный вывод

Рассмотрим тот же пример:

```
Z:\>python greeting.py
```

```
Введите ваше имя:Alexander
```

```
Введите вашу фамилию:Kononov
```

```
Привет, Alexander Kononov
```

```
Z:\>python greeting.py > output.txt
```

```
Alexander
```

```
Kononov
```

```
Z:\>
```

В текущей папке появится файл `output.txt` со следующим содержимым:

```
Введите ваше имя:Введите вашу фамилию:Привет, Alexander Kononov
```

Стандартные потоки ввода-вывода (6) — стандартный вывод

Когда мы перенаправили вывод в файл `output.txt`, то, что программа до этого выводила на экран (строчку Введите ваше имя: и т.д.), она стала выводить в файл — свои имя и фамилию я писал практически наугад, не видя, что программа меня спрашивает.

Дело в том, что в обоих случаях программа пишет в поток стандартного вывода `stdout`, который, однако, в первом случае был связан с экраном, и мы видели, что программа пишет, а во втором случае — с файлом, и на экран в результате ничего не выводится (зато весь вывод попадает в файл).

Стандартные потоки ввода-вывода (7) — стандартный вывод ошибок

Стандартный поток ошибок перенаправляют редко, для перенаправления используются похожие команды, но перед знаками > и >> нужно указать цифру 2:

```
program arg1 arg2 ... 2>errors.txt
```

```
program arg1 arg2 ... 2>>errors.txt
```

Описанный выше синтаксис связывания стандартных потоков с файлами одинаково работает и в Windows, и в unix-подобных системах (macOS и Linux).

Стандартные потоки ввода-вывода (8) — конвейер

Связывать стандартные потоки можно не только с файлами, но и между собой. Если у нас есть две программы, одна из которых (назовём её `prog1`) пишет данные на `stdout`, а вторая (назовём её `prog2`) читает данные со стандартного ввода (`stdin`), то их можно связать в *конвейер*:

```
prog1 | prog2
```

Стандартный вывод `stdout` программы `prog1` будет связан со стандартным вводом `stdin` программы `prog2`, то, всё, что будет выводить первая программа, будет считывать вторая программа.

Стандартные потоки ввода-вывода (9) — конвейер

В конвейере может быть сколько угодно команд:

```
prog1 | prog2 | prog3 | prog4
```

Здесь у команд `prog2` и `prog3` связаны и стандартные вводы, и стандартные выходы.

В командной строке unix-подобных операционных систем имеется довольно много встроенных команд, предназначенных для обработки текстовых данных (выбор подстрок, сортировка строчек и т.д.), эти программы удобно комбинировать при помощи конвейеров. Командная строка Windows гораздо беднее на эти возможности.

Стандартные потоки ввода-вывода (10) — конвейер

Возьмём текстовый файл `jack.txt` следующего содержания:

```
This is the house that Jack built.  
This is the malt  
That lay in the house that Jack built.  
This is the rat, that ate the malt  
That lay in the house that Jack built.  
This is the cat,  
That chased the rat, that ate the malt  
That lay in the house that Jack built.  
This is the dog, that worried the cat,  
That chased the rat, that ate the malt  
That lay in the house that Jack built.  
This is the cow with the crumpled horn,  
That tossed the dog, that worried the cat,  
That chased the rat, that ate the malt  
That lay in the house that Jack built.
```

...

Стандартные потоки ввода-вывода (11) — конвейер

...

This is the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog, that worried the cat,
That chased the rat, that ate the malt
That lay in the house that Jack built.
This is the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog, that worried the cat,
That chased the rat, that ate the malt
That lay in the house that Jack built.

...

Стандартные потоки ввода-вывода (12) — конвейер

...

This is the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog, that worried the cat,
That chased the rat, that ate the malt
That lay in the house that Jack built.

...

Стандартные потоки ввода-вывода (13) — конвейер

...

This is the cock that crowed in the morn,
That waked the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog, that worried the cat,
That chased the rat, that ate the malt
That lay in the house that Jack built.

...

Стандартные потоки ввода-вывода (14) — конвейер

...

This is the farmer sowing his corn,
That kept the cock that crowed in the morn,
That waked the priest all shaven and shorn,
That married the man all tattered and torn,
That kissed the maiden all forlorn,
That milked the cow with the crumpled horn,
That tossed the dog, that worried the cat,
That chased the rat, that ate the malt
That lay in the house that Jack built.

Стандартные потоки ввода-вывода (14) — `grep` и `find`

Для того, чтобы найти в нём все строки, в которых упоминается некоторая строчка, на Windows нужно использовать команду `find "искомая строка"` (кавычки обязательны), в unix-подобных системах — `grep "искомая строка"` (кавычки нужны, если в искомой строке есть пробелы).

Стандартные потоки ввода-вывода (15) — grep и find

Для примера, найдём в файле слово dog:

```
Z:\>find "dog" < jack.txt
```

```
This is the dog, that worried the cat,  
That tossed the dog, that worried the cat,  
That tossed the dog, that worried the cat,  
That tossed the dog, that worried the cat,  
That tossed the dog, that worried the cat,  
That tossed the dog, that worried the cat,  
That tossed the dog, that worried the cat,
```

```
Z:\>
```

Стандартные потоки ввода-вывода (16) — grep и find

Программа `find` считывает строки со стандартного ввода и выводит их на стандартный вывод. Аналогично работает и `grep`:

[illegible]

Стандартные потоки ввода-вывода (17) — sort

Команда `sort` (есть и там, и там) сортирует строки, считанные из `stdin`, и выводит их в алфавитном порядке на `stdout` (часть вывода опущена):

```
Z:\>sort < jack.txt
```

```
That chased the rat, that ate the malt
```

```
...
```

```
This is the cat,
```

```
This is the cock that crowed in the morn,
```

```
This is the cow with the crumpled horn,
```

```
This is the dog, that worried the cat,
```

```
This is the farmer sowing his corn,
```

```
This is the house that Jack built.
```

```
This is the maiden all forlorn,
```

```
This is the malt
```

```
This is the man all tattered and torn,
```

```
This is the priest all shaven and shorn,
```

```
This is the rat, that ate the malt
```


Стандартные потоки ввода-вывода (18) — sort

```
mazdaywik@Mazdaywik-NB10:~/modules$ sort < jack.txt
That chased the rat, that ate the malt
...
This is the cat,
This is the cock that crowed in the morn,
This is the cow with the crumpled horn,
This is the dog, that worried the cat,
This is the farmer sowing his corn,
This is the house that Jack built.
This is the maiden all forlorn,
This is the malt
This is the man all tattered and torn,
This is the priest all shaven and shorn,
This is the rat, that ate the malt
mazdaywik@Mazdaywik-NB10:~/modules$
```

Стандартные потоки ввода-вывода (18) — конвейер

Найдём все строки, содержащие слово `This`, отсортирует их по алфавиту и результат сложим в файл с именем `this.txt`:

```
Z:\>find "This" < jack.txt | sort > this.txt
```

```
mazdaywik@Mazdaywik-NB10:~/modules$ grep This < jack.txt | sort
```

Стандартные потоки ввода-вывода (19) — конвейер

В обоих случаях получим файл `this.txt` со следующим содержимым:

```
This is the cat,  
This is the cock that crowed in the morn,  
This is the cow with the crumpled horn,  
This is the dog, that worried the cat,  
This is the farmer sowing his corn,  
This is the house that Jack built.  
This is the maiden all forlorn,  
This is the malt  
This is the man all tattered and torn,  
This is the priest all shaven and shorn,  
This is the rat, that ate the malt
```

Стандартные потоки ввода-вывода (20)

Как видно, можно выполнить достаточно сложную задачу (выбрать из файла строки, содержащие некоторую подстроку, отсортировать их по алфавиту и записать результат в новый файл), скомбинировав вызовы нескольких встроенных команд операционной системы при помощи перенаправления стандартных потоков.

В обоих случаях у программы поиска подстроки (`find` или `grep`) стандартный ввод связывался с исходным файлом, стандартный вывод — со следующей программой в конвейере — командой сортировки `sort`. Её стандартный вывод перенаправлялся в целевой файл.

Стандартные потоки ввода-вывода (21) — аргументы

Для программ, работающих в командной строке, принято соглашение:

- ▶ Если среди аргументов командной строки имена файлов не указаны, то чтение осуществляется со стандартного ввода.
- ▶ Если имена файлов указаны, что обрабатываются все перечисленные файлы.
- ▶ Результат работы программы выводится на стандартный вывод.

Стандартные потоки ввода-вывода (22) — аргументы

Таким образом, командам `grep`, `find`, `sort` и другим встроенным командам можно в командной строке указывать и имена файлов:

```
sort input.txt
```

```
grep 'какая-то строка' one.txt two.txt three.txt
```

```
find "какая-то строка" one.txt two.txt three.txt
```

Стандартные потоки ввода-вывода (23) — псевдофайлы

В операционных системах предусмотрены некоторые псевдофайлы со специальными именами, которые не соответствуют реальным данным на диске, чтение и запись с ними особым образом обрабатываются операционной системой.

На Windows наиболее употребительными являются файлы с именами `nul` и `con`. Чтение из файла `nul` сразу же приводит к обнаружению конца файла (т.е. этот файл воспринимается как пустой), запись в него игнорируется. Чтение из файла `con` соответствует чтению с клавиатуры (даже если стандартный ввод перенаправлен), запись в него соответствует записи на экран (даже если перенаправлен стандартный вывод). Файлы `nul` и `con` неявно присутствуют в каждой папке.

Стандартные потоки ввода-вывода (24) — псевдофайлы

На unix-подобных системах чаще всего используется файл `/dev/null`, аналогичный файлу `nul` на Windows: при чтении он воспринимается как пустой (сразу же встречаем конец файла), при записи записываемые данные игнорируются.

Файлы `nul` и `/dev/null` используются в командной строке для того, чтобы подавить вывод на экран:

```
C:\...>program args ... > nul
```

```
user@comp:~/ ... $ program args ... > /dev/null
```


Стандартные потоки ввода-вывода (25) — псевдофайлы

С псевдофайлом con мы работали, когда создавали текстовый файл из командной строки на Windows:

```
C:\Users\Mazdaywik\New folder>copy con "new file.txt"
```

```
first line
```

```
second line
```

```
third line
```

```
^Z
```

```
Скопировано файлов:           1.
```

```
C:\Users\Mazdaywik\New folder>
```

Стандартные потоки ввода-вывода (26) — псевдофайлы

Команда `cat` в unix-системах читает указанные в командной строке файлы и выводит их содержимое на стандартный вывод. Если файлы не указаны, то она выводит на стандартный вывод содержимое стандартного ввода. Когда мы создавали файл в командной строке Linux, мы просто перенаправляли стандартный вывод:

```
mazdaywik@Mazdaywik-NB10:~/New folder$ cat > "new file.txt"
first line
second line
third line
mazdaywik@Mazdaywik-NB10:~/New folder$
```

Взаимодействие программ на Python с операционной системой

Модуль sys (1)

Для взаимодействия с операционной системой используется модуль `sys`, подключаемый командной

```
import sys
```

Чтобы посмотреть перечень всех доступных средств, нужно в среде IDLE ввести следующую команду:

```
>>> import sys  
>>> help(sys)
```

Модуль `sys` (2)

Нам все эти средства не нужны, кроме двух основных:

- ▶ доступ к стандартным потокам ввода-вывода,
- ▶ доступ к аргументам командной строки.

Стандартные потоки ввода-вывода доступны как переменные

- ▶ `sys.stdin`
- ▶ `sys.stdout`
- ▶ `sys.stderr`

В этих переменных находятся объекты файлов, связанные со стандартными потоками. Их можно использовать в файловых операциях точно также, как и объекты файлов, полученные из функции `open()`.

Модуль `sys` (3) — стандартные потоки

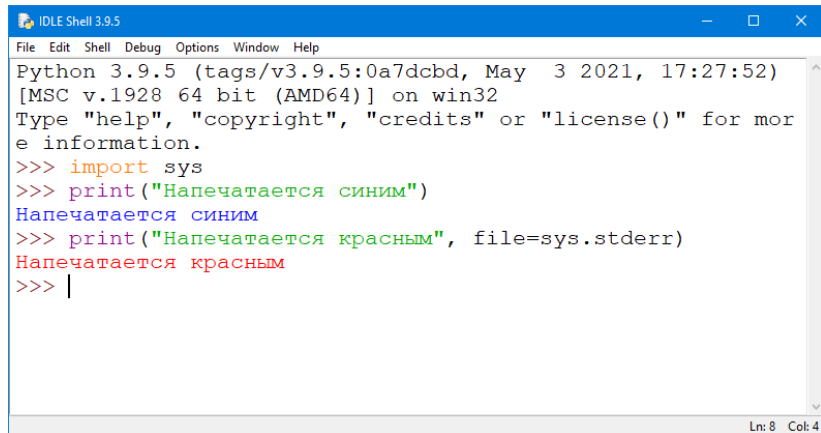
При запуске программ в среде IDLE конец файла для `sys.stdin` вводится как Ctrl-D в новой строке независимо от операционной системы, вывод на `sys.stdout` отображается синим цветом, на `sys.stderr` — красным цветом.

Пример: используем `.readlines()` для `sys.stdin`.

```
>>> sys.stdin.readlines()
one
two
three
>>> ['one\n', 'two\n', 'three\n']
```

Модуль sys (4)

Пример: выводим на `sys.stderr`:



```
Python 3.9.5 (tags/v3.9.5:0a7dcbbd, May 3 2021, 17:27:52)
[MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>> import sys
>>> print("Напечатается синим")
Напечатается синим
>>> print("Напечатается красным", file=sys.stderr)
Напечатается красным
>>> |
```

Ln: 8 Col: 4

Модуль `sys` (5)

Мы говорили о том, что функция `print()` по умолчанию пишет на экран, чтобы перенаправить вывод в файл, мы использовали параметр `file=`. Если быть точным, функция `print()` всегда пишет в файл, по умолчанию `file=sys.stdout`.

Модуль `sys` (6) — аргументы командной строки

Для доступа к аргументам командной строки используется переменная `sys.argv`, в которой находится список аргументов. Этот список всегда непустой, т.к. значением нулевого элемента списка `sys.argv[0]` является имя самой запущенной программы. Для примера создадим файл `test-argv.py` со следующим содержимым:

```
import sys

for i, arg in enumerate(sys.argv):
    print(i, arg)
```

Модуль sys (7) — аргументы командной строки

Запустим его:

```
Z:\>python test-argv.py one two three
```

```
0 test-argv.py
```

```
1 one
```

```
2 two
```

```
3 three
```

```
Z:\>
```

```
mazdaywik@Mazdaywik-NB10:~/modules$ python3 test-
```

```
argv.py one two three
```

```
0 test-argv.py
```

```
1 one
```

```
2 two
```

```
3 three
```

Модуль sys (8) — аргументы командной строки

Действительно, нулевой аргумент — имя запущенной программы, остальные — параметры, указанные после имени.

Если параметр содержит пробелы, его нужно заключить в кавычки:

```
Z:\>python test-argv.py "with space" without space
0 test-argv.py
1 with space
2 without
3 space
```

Пробел между словами `with space` стал частью параметра, т.к. внутри кавычек, пробел между словами `without` и `space` стал разделителем параметров.

Модуль `sys` (9) — аргументы командной строки

Пример. Напишем программу `cat.py`, имитирующую работу программы `cat` в unix-системах. Программа `cat` читает файлы, перечисленные в командной строке, каждый из них открывает и выводит его содержимое на стандартный вывод. Если имена файлов отсутствуют, то читается стандартный ввод.

```
import sys

def main():
    if len(sys.argv) == 1:
        process_stream(sys.stdin)
    else:
        for name in sys.argv[1:]:
            with open(name) as stream:
                process_stream(stream)
```

Модуль sys (10) — аргументы командной строки

```
def process_stream(stream):  
    sys.stdout.write(stream.read())
```

```
if __name__ == "__main__":  
    main()
```

Модуль `sys` (11) — аргументы командной строки

Программа написана в рекомендуемом стиле для консольных программ.

В программе объявлены две функции `main()` и `process_stream()`, в самом конце вызывается функция `main()`. Условие `__name__ == "__main__"` выполняется всегда при обычном запуске программы.

За счёт того, что основной код программы сосредоточен в функции `main()`, а не «размазан» по всему исходнику, понимание программы упрощается, кроме того, в основном коде мы можем вызывать функции до их объявления.

Функция `main()` описывает обычную логику для консольных программ: обрабатывать либо файлы (если указаны явно), или стандартный ввод. Поскольку обработка и того, и другого выполняется одинаково, эта логика вынесена во вспомогательную функцию `process_stream()`.

Модуль sys (12) — аргументы командной строки

Убедимся, что программа работает, как мы ожидаем:

```
mazdaywik@Mazdaywik-NB10:~/modules$ python3 cat.py test-  
argv.py this.txt
```

```
import sys
```

```
for i, arg in enumerate(sys.argv):  
    print(i, arg)
```

```
This is the cat,
```

```
This is the cock that crowed in the morn,
```

```
This is the cow with the crumpled horn,
```

```
This is the dog, that worried the cat,
```

```
This is the farmer sowing his corn,
```

```
This is the house that Jack built.
```

```
This is the maiden all forlorn,
```

```
This is the malt
```

```
This is the man all tattered and torn,
```

```
This is the priest all shaven and shorn,
```

```
This is the rat, that ate the malt
```

Модуль sys (13) — аргументы командной строки

```
mazdaywik@Mazdaywik-NB10:~/modules$ cat  
hello!  
hello!  
mazdaywik@Mazdaywik-NB10:~/modules$ python3 cat.py  
hello!  
hello!  
mazdaywik@Mazdaywik-NB10:~/modules$
```

Мы видим, что поведение встроенной программы cat и написанной нами cat.py идентично на Linux.

Модуль sys (14) — аргументы командной строки

Кроме того, программа cat.py будет работать и на Windows:

```
Z:\>python cat.py test-argv.py this.txt
```

```
import sys
```

```
for i, arg in enumerate(sys.argv):
```

```
    print(i, arg)
```

```
This is the cat,
```

```
This is the cock that crowed in the morn,
```

```
This is the cow with the crumpled horn,
```

```
This is the dog, that worried the cat,
```

```
This is the farmer sowing his corn,
```

```
This is the house that Jack built.
```

```
This is the maiden all forlorn,
```

```
This is the malt
```

```
This is the man all tattered and torn,
```

```
This is the priest all shaven and shorn,
```

```
This is the rat, that ate the malt
```

Модуль sys (15) — аргументы командной строки

Пример. Напишем программу `grep.py`, имитирующую работу встроенной программы `grep` unix-подобных систем. Программа принимает в качестве первого параметра искомую подстроку, после которой может следовать несколько имён файлов. Программа выводит все строки указанных файлов, содержащие данную подстроку. Если указано два и более файлов, то в начало выводимой строки добавляется имя сканируемого файла. Если файлы не указаны, читается стандартный ввод. Программа может принимать ключ `-v`, говорящий о том, что вывод нужно инвертировать, т.е. выводить строки, *не содержащие* указанной подстроки.

(На самом деле, сходство будет неточное, т.к. программа `grep` принимает *шаблон*, а наша программа — подстроку. Кроме того, ключей у команды `grep` на самом деле очень много, а у нас будет только один `-v`.)

Модуль sys (16) — аргументы командной строки

Для простоты будем считать, что ключ `-v` указывается перед искомой подстрокой.

```
import sys

def main():
    v = False
    argv = sys.argv[1:]
    if len(argv) > 0 and argv[0] == '-v':
        v = True
        argv = argv[1:]

    if len(argv) == 0:
        print(sys.argv[0]
              + ': Не указана строка поиска',
              file=sys.stderr)
    return
```

Модуль sys (17) — аргументы командной строки

```
...
search_string = argv[0]
files = argv[1:]

if len(files) == 0:
    process(search_string, sys.stdin, None, v)
elif len(files) == 1:
    with open(files[0]) as stream:
        process(search_string, stream, None, v)
else:
    for name in files:
        with open(name) as stream:
            process(search_string, stream, name, v)
```

Модуль sys (18) — аргументы командной строки

```
def process(search_string, stream, name, v):  
    if name != None:  
        prefix = name + ':'  
    else:  
        prefix = ''
```

Модуль sys (19) — аргументы командной строки

```
for line in stream:
    found = (search_string in line)
    if not v:
        if found:
            print(prefix + line, end='')
    else: # -v указан
        if not found:
            print(prefix + line, end='')
```

```
if __name__ == "__main__":
    main()
```

Модуль `sys` (20) — аргументы командной строки

Функция `process` принимает, помимо потока, который нужно просканировать, искомую строку, имя файла и факт наличия параметра `-v`. Переменная `prefix` содержит строку, добавляемую в начало вывода каждой строки: она пустая, если имя файла печатать не надо (сканируется стандартный ввод или файл один), в ней имя файла и двоеточие в остальных случаях.

Модуль sys (21) — аргументы командной строки

Код функции process можно несколько сократить:

```
def process(search_string, stream, name, v):  
    if name  $\neq$  None:  
        prefix = name + ':'  
    else:  
        prefix = ''  
  
    for line in stream:  
        found = (search_string in line)  
        if found  $\neq$  v:  
            print(prefix + line, end='')
```


Модуль sys (22) — аргументы командной строки

Можно убедиться, что написанная программа правильно повторяет поведение встроенной команды `grep`:

```
mazdaywik@Mazdaywik-NB10:~/modules$ grep if cat.py grep.py
cat.py:    if len(sys.argv) == 1:
cat.py:if __name__ == "__main__":
grep.py:    if len(argv) > 0 and argv[0] == '-v':
grep.py:    if len(argv) == 0:
grep.py:    if len(files) == 0:
grep.py:    elif len(files) == 1:
grep.py:    if name != None:
grep.py:        if found != v:
grep.py:if __name__ == "__main__":
```

Модуль sys (23) — аргументы командной строки

```
mazdaywik@Mazdaywik-NB10:~/modules$ python3 grep.py if cat.py
cat.py:    if len(sys.argv) == 1:
cat.py:if __name__ == "__main__":
grep.py:    if len(argv) > 0 and argv[0] == '-v':
grep.py:    if len(argv) == 0:
grep.py:    if len(files) == 0:
grep.py:    elif len(files) == 1:
grep.py:    if name != None:
grep.py:        if found != v:
grep.py:if __name__ == "__main__":
```

Модуль sys (24) — аргументы командной строки

```
mazdaywik@Mazdaywik-NB10:~/modules$ grep for test-argv.py
for i, arg in enumerate(sys.argv):
mazdaywik@Mazdaywik-NB10:~/modules$ grep -v for test-argv.py
import sys

    print(i, arg)
mazdaywik@Mazdaywik-NB10:~/modules$ python3 grep.py for test-
for i, arg in enumerate(sys.argv):
mazdaywik@Mazdaywik-NB10:~/modules$ python3 grep.py -v for te
import sys

    print(i, arg)
mazdaywik@Mazdaywik-NB10:~/modules$
```