

---

# Лаба № 1. Настройка компьютера

## Содержание

Для начала .....	1
Запуск терминала .....	1
Установка интерпретатора Python .....	1
Работа с файловой системой .....	2
Директории .....	3
Файлы .....	6
Загрузка задания .....	11
Установка текстового редактора .....	11
Sublime Text 3 .....	12
Первый взгляд на файл-заготовку .....	13
Запуск доктестов .....	14

## Для начала

### Запуск терминала

Терминал — это программа, позволяющая взаимодействовать с компьютером через ввод команд. В этом курсе, вне зависимости от операционной системы твоего компьютера (Windows, MacOS, Linux), терминал будет необходимым инструментом.



Если у тебя MacOS или любой Linux (типа Ubuntu), то программа `Terminal` уже установлена. Открывай ее скорее и приступай!



Пользователям Windows рекомендуется загрузить и установить терминал, называемый [GitBash](#)<sup>1</sup>.

### Установка интерпретатора Python

Python нужно взять [здесь](#)<sup>2</sup>. Загрузи один из дистрибутивов (например, «Windows x86-64 MSI installer» или «Mac OS X 64-bit installer»). Если операционная система

---

<sup>1</sup> <https://github.com/msysgit/msysgit/releases/download/Git-1.9.5-preview20150319/Git-1.9.5-preview20150319.exe>

<sup>2</sup> <https://www.python.org/downloads/release/python-342>

64-битная, нужно выбирать 64-битный дистрибутив. (Если твоему компьютеру меньше 2-х лет, то скорее всего он 64-битный. Если определение битности вызывает недоумение — задай вопрос.)



Отдельно для пользователей Windows подчеркнем, что при установке нужно выбрать опцию «Add python.exe to Path» как показано на рисунке.



## Работа с файловой системой

Первым делом запусти терминал, если ты этого еще не сделал. Здесь и далее предполагается, что пользователи Windows пользуются установленным [GitBash<sup>3</sup>](http://git-scm.com/downloads), пользователи Linux и Mac пользуются стандартными терминалами.

---

<sup>3</sup> <http://git-scm.com/downloads>



С самого начала ты находишься в *домашней папке*. Об этом факте как бы сообщает нам символ `~` (тильда). Также можно заметить, что в терминале отображается имя пользователя и название компьютера.



Поскольку терминал GitBash эмулирует работу терминала Unix/Linux, то отображаемая в GitBash домашняя папка не совпадает с домашней папкой пользователя Windows, а является также эмулируемой.

Теперь настало время изучить и уяснить как работает файловая система. Перемещать, копировать, открывать и удалять файлы с помощью мышки в системах с графическим интерфейсом (то есть с «окошечками») умеют все. Операции с файлами и папками в терминале происходят другим образом. Тебе даже не встретится, горячо любимая *обычными пользователями*, иконка «Мой компьютер».

Ничего страшного, сейчас мы изучим как работать с файловой системой с помощью командной строки (то есть терминала). В принципе, все, что можно сделать визуальным образом (мышкой в красивых «окошечках»), можно сделать и текстовыми командами.

## Директории

Прежде всего познакомимся с нашим другом, `ls` (сокращение от list, то есть список).

Команда `ls` выводит список всех файлов в текущем директории. Директорий, кстати, это более правильное название для папки, такой, например, как «Мои документы». В терминале существует понятие текущего директория, то есть того

директория в котором ты находишься в текущий момент. При запуске терминала текущим директориум становится домашний (home) директориум пользователя от имени которого запущен терминал. То есть если прямо сейчас выполнить команду `ls`, то она выведет содержание твоего домашнего директориума.

Набери команду `ls`:

```
.....
student ~ # ls
student ~ #
.....
```

Странно. Ничего вроде бы не произошло. Но это и правильно — сейчас в домашнем директориуме ничего нет. Значит нужно туда что-нибудь добавить!

### Создание директориумов

Вот еще одна полезная команда `mkdir` (сокращение от make directory, то сделать директориум). Она нужна для создания новых директориумов. В отличие от `ls`, ввести название и нажать **Enter** в этом случае недостаточно. Нужно также указать название нового создаваемого директориума. Безграничная степень внутренней дисциплины и самоорганизации заставляет нас создать директориум для этой лабораторной работы с названием `lab_01`:

```
.....
student ~ # mkdir lab_01
student ~ #
.....
```



В случае, когда после названия команды следуют какие-либо уточнения (как, например, название директориума), говорят что команда вызвана с одним или несколькими параметрами. Не все команды требуют ввода дополнительного параметра/параметров. Некоторые команды (например `ls`) имеют множество необязательных параметров, которые можно не указывать.

Итак, директориум создан. Нужно проверить, что новый директориум действительно существует:

```
.....
student ~ # ls
lab_01
student ~ #
.....
```

Вот он — новый директорию. Крутотень!

## Изменение текущего директория

Для изменения текущего директория используется команда `cd` (сокращение от `change directory`).

Эта команда ожидает параметр в виде названия целевого директория для перехода. Зайди в директорию `lab_01` с помощью команды:

```
student ~ # cd lab_01
student ~/lab_01 #
```

Знак `~` (тильда) превратился в `~/lab_01`. Это означает, что текущий директорию изменился на `lab_01`.

Внутри директория `lab_01` еще ничего нет, ни файлов, ни вложенных директориев. Удостовериться в этом можно с помощью уже известной команды `ls`. Результат ее выполнения как бы скажет нам: «Директорию пуст».

Теперь предположим, что нужно вернуться в домашний директорию `~`. Это можно сделать несколькими способами.

Первый способ:

```
student ~/lab_01 # cd ..
student ~ #
```



Две точки в Unix/Linux означают внешний по отношению к текущему директории. То есть команда `cd ..` приводит к переходу в *родительский* директорию из *дочернего*.

Второй способ:

```
student ~/lab_01 # cd
student ~ #
```

Ввод команды `cd` без параметров эквивалентен переходу в домашний директорию. Бывает удобно использовать эту команду для быстрого перехода в домашний директорию из длинной последовательности вложенных директориев

(ну вот, например, отсюда `/home/student/somewhere/deep/inside/the/hell`).

## Удаление директориев

Раз директории можно создавать, то можно их и удалять. И это достаточно частая ситуация, не правда ли? Для удаления директория используй команду `rm -r`. Параметр `-r` в данном случае означает, что удаление должно произойти рекурсивно, то есть со всем содержимым.

Подобно команде `mkdir`, команда `rm -r` требует указания параметра с именем удаляемого директория.

Сделай вот что:

1. Создай директорий с названием `my_folder`.
2. С помощью команды `ls` удостоверься, что директорий создан.
3. Удали директорий командой `rm -r my_folder`.
4. Еще раз запусти `ls`, чтобы проверить, что на этот раз директорий `my_folder` отсутствует.

## Файлы

Было проделано уже достаточно много, однако все действия касались директориев. Сами по себе директории прекрасны, но кроме имени не несут в себе информации. Настало время наполнить их чем-то более содержательным. Файлами.

Прежде всего научимся создавать файлы. Разница между файлами и директориями в принципе очевидна. Более чем полностью сомнения могут быть развеяны попыткой «войти» в, например, текстовый файл командой `cd`.

Попробуй создать текстовый файл, содержащий предложение: «Я сдам сессию на отлично!» Это можно сделать множеством способов. Для начала попробуй воспользоваться командой `echo` (эхо).

Команда `echo` выводит в терминал то, что передается ей в качестве параметров.

```
.....
student ~ # echo Превед
Превед
student ~ # echo Хватит повторять
```

```
Хватит повторять
student ~ # echo Я тебе говорю!
Я тебе говорю!
```

---

Говорят, что то, что выводит компьютер после ввода команды и нажатия **Enter** называется *выводом* (output) этой команды, а то, что передавалось в качестве параметров называют *ввод* (input).

### Создание файлов

В Unix/Linux для создания файла используют команду `touch`. Например, для создания файла `my_file` нужно сделать следующее:

```
student ~ # touch my_file
student ~ # ls
lab_01 my_file
```

---

Это было не сложно! Для того, чтобы заглянуть внутрь файла понадобится другая команда — `cat`.

```
student ~ # cat my_file
student ~ #
```

---

Нетрудно сообразить, что пустой вывод команды `cat` как бы говорит нам, что файл, созданный командой `touch`, ничего не содержит.

Для удаления файла можно воспользоваться уже знакомой командой `rm`. На этот раз без параметра `-r`.

```
student ~ # ls
lab_01 my_file
student ~ # rm my_file
student ~ # ls
lab_01
```

---



Используй команду `rm` с аккуратностью! Эта команда по-настоящему удаляет файлы, а не перемещает их в корзину, как ты, быть может, привык под Windows/MacOS. Обратить действие команды `rm` простыми способами невозможно. Так что подумай дважды или даже трижды!

Директории ты создавал и удалял. С файлами можно делать гораздо больше.

Создай новый файл (ты ведь только что удалил, то что было создано).

```
student ~ # touch my_file
student ~ # ls
lab_01 my_file
```

Добавь текст внутрь файла. Для этого можно использовать команду `echo`:

```
student ~ # echo "Я сдам сессию на отлично!" > my_file
```

Для тех кому интересно, символ `>` означает перенаправление с экрана терминала в файл, указанный после этого знака. В приведенном примере текст «Я сдам сессию на отлично!» попадает в файл `my_file`.



Использовать перенаправление нужно с аккуратностью — все что находилось в файле будет заменено на новый текст. Для того чтобы добавлять текст к концу существующего файла, применяй `>>`.

## Копирование файлов

Теперь представим, что тебе надо скопировать файл. Для этого предназначена команда `cp` (сору, копия). Она ожидает передачи ей двух аргументов: первый — название файла, который необходимо скопировать, второй — имя нового файла, в который необходимо скопировать исходный файл.

Например, для копирования `my_file` в `new_file` нужно сделать так:

```
student ~ # cp my_file new_file
student ~ # ls
lab_01 my_file new_file
```

Если заглянуть в каждый из этих файлов с помощью `cat`, можно обнаружить, что `new_file` является копией исходного файла `my_file`. Как раз то, что было нужно:

```
student ~ # cat new_file
Я сдам сессию на отлично!
```



Зачастую нужно скопировать файл из одного директория в другой. Команда `cp` может принимать параметры содержащие также и пути к файлам. Например, `aaa/bbb/ccc.txt` это путь к файлу, содержащему *некоторый текст*.

Можно сделать что-то вроде этого:

```
student ~ # cp aaa/bbb/ccc.txt ccc.txt
student ~ # ls
lab_01 my_file new_file ccc.txt
```



Если при копировании нет потребности изменять имя файла, то можно использовать точку (`.`) в качестве второго параметра. Результат при этом будет совершенно тот же:

```
student ~ # cp aaa/bbb/ccc.txt .
student ~ # ls
lab_01 my_file new_file ccc.txt
```

В Unix/Linux точка `.` означает сокращение для «текущий директорий». То есть команда `cp` в данном случае означает: скопируй файл `aaa/bbb/ccc.txt` в текущий директорий.

Похожим образом можно скопировать `ccc.txt` в директорий `lab_01`.

```
student ~ # cp aaa/bbb/ccc.txt lab_01
student ~ # ls lab_01
ccc.txt
```

## Перемещение файлов

При копировании файла создается новый файл полностью соответствующий исходному, который не претерпевает никаких изменений. Зачастую бывает необходимо, чтобы файл переместился, то есть чтобы его копия возникла в новом месте, а в старом файл был бы удален.

Конечно можно было бы воспользоваться связкой команд `cp` и `rm`, но есть более удобный способ — команда `mv` (move, переместить), принимающая два аргумента:

```
student ~ # mv new_file lab_01
student ~ # ls
```

```
lab_01 my_file
student ~ # cd lab_01
student ~/lab_01 # ls
new_file
```

---

Файл `new_file` был перемещен в директорию `lab_01`. Видно, что директорию `lab_01` находится в домашней директории, там, где располагался до выполнения команды файл `new_file`. Для успешного выполнения команды необходимо, чтобы директорию `lab_01` существовал в текущей директории, иначе это могло привести бы к переименованию файла (об этом ниже).

Теперь представим, что нужно переместить файл обратно в домашнюю директорию. Для этого можно указать команде `mv`, что переместить файл нужно в родительскую директорию `..`:

---

```
student ~/lab_01 # ls
new_file
student ~/lab_01 # mv new_file ..
student ~/lab_01 # ls
student ~/lab_01 # cd
student ~ # ls
new_file
```

---

Только что файл `new_file` был перемещен обратно в домашнюю директорию.

## Переименовывание файлов

И последнее, можно переименовать файл. Для переименований используется уже рассмотренная команда `mv`. В этом случае команде нужно также передать два аргумента: во-первых, имя файла для переименования, во-вторых, новое название файла:

---

```
student ~/lab_01 # mv new_file best_name_ever
student ~/lab_01 # ls
best_name_ever
```

---



Конечно выводимая компьютером информация будет на английском языке, который тебе, может быть, не очень хорошо знаком. С этим надо что-то делать! Английский язык в области информационных технологий — совершенно необходимая вещь. Это как латынь для медиков. Без знания английского

языка очень тяжело стать настоящим профессионалом. Если есть трудности — немедленно запишись на дополнительные курсы — это очень важно!

## Загрузка задания

Теперь настало время загрузить файл-заготовку к этой лабораторной работе `lab_01.py`<sup>4</sup>. Для этого можно воспользоваться командой `curl`, которая позволяет загружать файлы из интернета.

---

```
student ~/lab_01 # curl -O http://cucp.bitbucket.org/lab_01.py
```

---

На экране должна отобразиться информация о загрузке. Проверить то, что файл теперь существует в текущей папке можно с помощью команды `ls`.



Пользователям Windows рекомендуется работать далее не в домашней папке, которая, как было сказано ранее, является эмулированной и находится в глубоких недрах файловой системы Windows, а создать отдельную папку (например на диске `C:`) и продолжить работу в ней.

---

```
student ~/lab_01 # cd
student ~ # mkdir /C/cucp
student ~ # cp -r 'lab_01' /C/cucp
student ~ # cd /C/cucp/lab_01
```

---

Не лишним также будет проверить, что загруженный файл скопировался

---

```
student /C/cucp/lab_01 # ls
lab_01.py
```

---

## Установка текстового редактора

Установленный ранее **интерпретатор Python** позволяет *выполнять* код. Но тебе также потребуется текстовый редактор, который существенно облегчит *написание* программ.

---

<sup>4</sup> [http://cucp.bitbucket.org/lab\\_01.py](http://cucp.bitbucket.org/lab_01.py)

Текстовый редактор — это как Microsoft Word для написания программ — он позволяет создавать, изменять и сохранять файлы с кодом.

Существует множество текстовых редакторов, каждый со своим набором особенностей. Для работы над заданиями курса ты можешь выбрать любой, однако надо помнить, что он должен:

- открывать и редактировать новые файлы простым способом;
- хорошо работать как с файлами Python'a, так и с неформатированным текстом;
- отображать номера строк.

Все удобные редакторы в дополнение к этому умеют:

- осуществлять подсветку синтаксических конструкций;
- выполнять команды с помощью комбинаций клавиш.

В этом курсе рекомендуется использовать редактор [Sublime Text 3](http://www.sublimetext.com/3)<sup>5</sup>.

## Sublime Text 3



Пожалуйста, пожалуйста, *пожалуйста* не используйте Microsoft Word для редактирования программ. Word предназначен для редактирования текстов на естественных языках (типа русском или английском). Если будешь использовать Word для написания программ — проблемы обязательно появятся!



Вместе с Python устанавливается редактор IDLE. Его использование **не рекомендуется**, поскольку в этом случае ты не научишься работать с командной строкой и не будешь контролировать того, что происходит. Очень важно запускать свои файлы из терминала с самого начала.

## Некоторые горячие клавиши

- **Ctrl+s** — сохранить текущий файл;
- **Ctrl+x** — забрать текущий абзац в буфер обмена;

---

<sup>5</sup> <http://www.sublimetext.com/3>

- **Ctrl+x** — вставить текст из буфера обмена;
- **Ctrl+d** — выбрать текущее слово, при повторном нажатии в область выбора попадет следующее место в тексте с таким словом (очень удобно для переименований);
- **Ctrl+z** — откатить одно действие;
- **Ctrl+y** — вернуть откатенное действие;
- **Ctrl+Tab** — выбрать следующую вкладку;
- **Ctrl+Shift+Tab** — выбрать предыдущую вкладку;
- **Ctrl+f** — поиск слова;
- **Ctrl+Shift+f** — поиск слова во всех вкладках;
- **Ctrl+Shift+p** — открывает небольшую панель в которую можно вводить команды, например, `ss python` активирует подсветку Python-кода, `reindent` перерасставит отступы в файле.



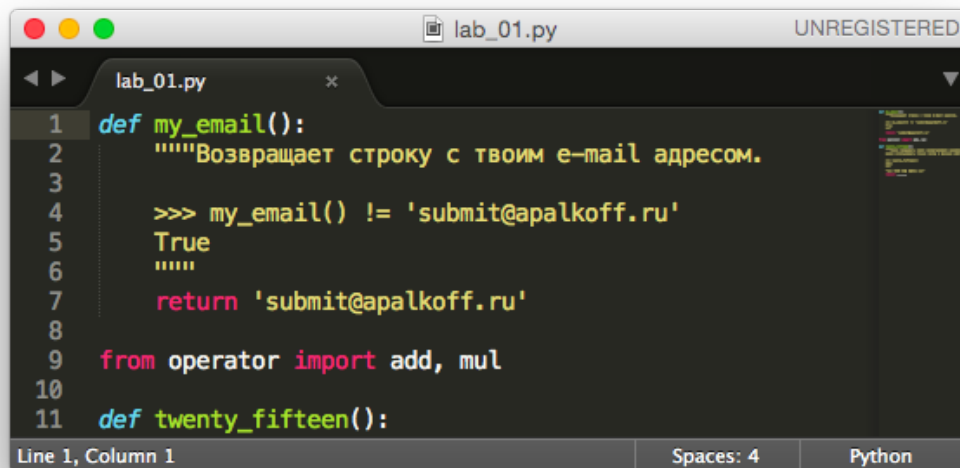
если есть что-то что ты хочешь от текстового редактора — Sublime вероятно может это сделать. Просто погугли!

## Первый взгляд на файл-заготовку

Обрати внимание на строки желтого, которые заключены в тройные двойные кавычки `"""`. Такой текст называют **докстринг** (*docstring*). В этом тексте описывается, что должна делать рассматриваемая функция.

В докстринге встречаются строки, начинающиеся с `>>>` — это **доктесты**. С помощью доктестов можно показать на конкретных примерах, как работает функция: «Если в интерпретатор ввести этот код (говорит нам строка с `>>>`), то следует ожидать такого результата (говорит нам строка следующая за `>>>`)».

В частности, рассмотрим функцию `my_email()`:



```
1 def my_email():
2     """Возвращает строку с твоим e-mail адресом.
3
4     >>> my_email() != 'submit@apalkoff.ru'
5     True
6     """
7     return 'submit@apalkoff.ru'
8
9 from operator import add, mul
10
11 def twenty_fifteen():
```

Доктест проверяет, что ты сменил выражение возврата с `submit@apalkoff.ru` на что-то другое.



Никогда не меняй доктесты в заданиях! Единственная часть, которую можно редактировать — это код. Отступление от этого правила приведет к тому, что количество баллов за задание будет равно нулю.

Как же запустить эти тесты? Хорошо что спросил!

## Запуск доктестов

Для запуска тестов в терминале следует набрать следующую команду:

```
python3 -m doctest lab_01.py
```



Если вдруг твой компьютер работает под операционной системой Windows и команда `python3` не работает, попробуй набирать `python` или `py`. Если и это не помогает, то скорее всего ошибка в установке переменной среды `PATH`. Если застрял — обратиться за помощью!

После запуска доктестов в терминале будет написано, что тесты не проходят.

```
1. apalkoff@mashabook: ~/prog/cucp.source/lab (zsh)
→ lab python3 -m doctest lab_01.py
*****
File "/Users/apalkoff/prog/cucp.source/lab/lab_01.py", line 4, in lab_01.my_email
Failed example:
    my_email() != 'submit@apalkoff.ru'
Expected:
    True
Got:
    False
*****
File "/Users/apalkoff/prog/cucp.source/lab/lab_01.py", line 15, in lab_01.twenty_fifteen
Failed example:
    twenty_fifteen()
Exception raised:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/doctest.py", line 1324, in __run
    compileflags, 1), test.globs)
  File "<doctest lab_01.twenty_fifteen[0]>", line 1, in <module>
    twenty_fifteen()
  File "/Users/apalkoff/prog/cucp.source/lab/lab_01.py", line 19, in twenty_fifteen
    return _____
NameError: name '_____' is not defined
*****
2 items had failures:
  1 of 1 in lab_01.my_email
  1 of 1 in lab_01.twenty_fifteen
***Test Failed*** 2 failures.
→ lab
```

Ой, ой — целых две ошибки! :( Перед их устранением посмотри более внимательно, что написано на экране, например, про `my_email`.

```
1. apalkoff@mashabook: ~/prog/cucp.source/lab (zsh)
*****
File "/Users/apalkoff/prog/cucp.source/lab/lab_01.py", line 4, in lab_01.my_email
Failed example:
    my_email() != 'submit@apalkoff.ru'
Expected:
    True
Got:
    False
*****
```

Видно, что ошибка находится в файле `lab_01.py` в строке `4`, в функции `my_email`. Теперь ты точно знаешь где искать баг! (Именно поэтому важно, чтобы текстовый редактор отображал номера строк).

Посмотри немного на строку `4`. А теперь давай разберемся о чем говорит тест: при вызове функции `my_email` без аргументов она должна **не** возвращать

строку `submit@apalkoff.ru`. Проблема в том, что возвращается как раз эта строка! Измени ее на свой e-mail.

```
def my_email():  
    """Возвращает строку с твоим e-mail адресом.  
  
    >>> my_email() != 'submit@apalkoff.ru'  
    True  
    """  
    return 'frosya@mail.ru'
```

После сохранения изменений в файле попробуй еще разок запустить доктесты:

```
python3 -m doctest lab_01.py
```

Тест для `my_email` теперь должен проходить. Один готов!

Во второй функции `twenty_fifteen` нужно заполнить пустоту после `return` выражением, значение которого будет равно 2015, используя только целые и функции `add` и `mul`. Это можно сделать множеством простых способов — попробуй изобрести что-нибудь оригинальное. После этого проверь, что тесты проходят.



Если все тесты проходят, Python не выведет сообщение об успехе. По умолчанию доктесты сообщают только о **провале**.

Однако, если очень хочется удостовериться, что все доктесты проходят, можно воспользоваться дополнительным параметром `'-v'`, который приведет к полному выводу информации, например, об успешном прохождении доктестов.

```
python3 -m doctest -v lab_01.py
```

После выполнения всех тестов на экране будет выведено что-то вроде.



```
1. apalkoff@mashabook: ~/prog/cucp...
→ lab python3 -m doctest -v lab_01.py
Trying:
    my_email() != 'submit@apalkoff.ru'
Expecting:
    True
ok
Trying:
    twenty_fifteen()
Expecting:
    2015
ok
1 items had no tests:
    lab_01
2 items passed all tests:
   1 tests in lab_01.my_email
   1 tests in lab_01.twenty_fifteen
2 tests in 3 items.
2 passed and 0 failed.
Test passed.
→ lab
```



Когда в файле-заготовке будут проходить все тесты, отправь его на адрес `submit@apalkoff.ru`, прикрепив к письму. Изменять название файла крайне не рекомендуется, поскольку в этом случае робот проигнорирует письмо. Файл нужно отправить до истечения установленного срока.