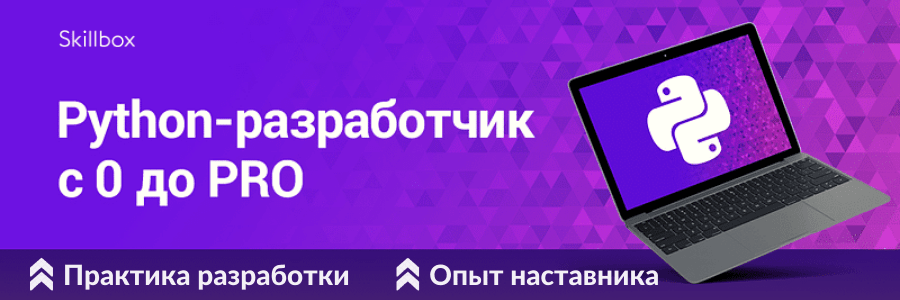
# **Работа с файлами в Python с помощью модуля OS**

12386

[](https://pythonru.com/go/python-developer-head)

Обработка файлов в Python с помощью модуля os включает создание, переименование, перемещение, удаление файлов и папок, а также получение списка всех файлов и каталогов и многое другое.

В индустрии программного обеспечения большинство программ тем или иным образом обрабатывают файлы: создают их, переименовывают, перемещают и так далее. Любой программист должен обладать таким навыком. С этим руководством вы научитесь использовать модуль os в Python для проведения операций над файлами и каталогами вне зависимости от используемой операционной системы.

Важно знать, что модуль os используется не только для работы с файлами. Он включает массу методов и инструментов для других операций: обработки переменных среды, управления системными процессами, а также аргументы командной строки и даже расширенные атрибуты файлов, которые есть только в Linux.

Модуль встроенный, поэтому для работы с ним не нужно ничего устанавливать.

## **Вывод текущей директории**

Для получения текущего рабочего каталога используется os.getcwd():

import os

# вывести текущую директорию

print("Текущая деректория:", os.getcwd())

os.getcwd() возвращает строку в Юникоде, представляющую текущий рабочий каталог. Вот пример вывода:

Текущая деректория: C:\python3\bin

## **Создание папки**

Для создания папки/каталога в любой операционной системе нужна следующая команда:

# создать пустой каталог (папку)

os.mkdir("folder")

После ее выполнения в текущем рабочем каталоге тут же появится новая папка с названием «folder».

[](https://pythonru.com/go/python-web-skills)

Если запустить ее еще раз, будет [вызвана ошибка](https://pythonru.com/osnovy/obrabotka-iskljuchenij-python-blok-try-except-blok-finally) FileExistsError, потому что такая папка уже есть. Для решения проблемы нужно запускать команду только в том случае, если каталога с таким же именем нет. Этого можно добиться следующим образом:

# повторный запуск mkdir с тем же именем вызывает FileExistsError,

# вместо этого запустите:

if not os.path.isdir("folder"):

os.mkdir("folder")

Функция os.path.isdir() вернет True, если переданное имя ссылается на существующий каталог.

## **Изменение директории**

Менять директории довольно просто. Проделаем это с только что созданным:

# изменение текущего каталога на 'folder'

os.chdir("folder")

Еще раз выведем рабочий каталог:

# вывод текущей папки

print("Текущая директория изменилась на folder:", os.getcwd())

Вывод:

Текущая директория изменилась на folder: C:\python3\bin\folder

## **Создание вложенных папок**

Предположим, вы хотите создать не только одну папку, но и несколько вложенных:

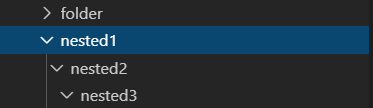
# вернуться в предыдущую директорию

os.chdir("..")

# сделать несколько вложенных папок

os.makedirs("nested1/nested2/nested3")

Это создаст три папки рекурсивно, как показано на следующем изображении:



## **Создание файлов**

Для [создания файлов в Python](https://pythonru.com/osnovy/fajly-v-python-vvod-vyvod) модули не нужны. Можно использовать встроенную функцию open(). Она принимает название файла, который необходимо создать в качестве первого параметра и желаемый режим открытия — как второй:

# создать новый текстовый файл

text\_file = open("text.txt", "w")

# запить текста в этот файл

text\_file.write("Это текстовый файл")

w значит write (запись), a — это appending (добавление данных к уже существующему файлу), а r — reading (чтение). Больше о режимах открытия можно почитать [здесь](https://pythonru.com/osnovy/fajly-v-python-vvod-vyvod" \l "-open).

## **Переименование файлов**

С помощью модуля os достаточно просто переименовать файл. Поменяем название созданного в прошлом шаге.

# переименовать text.txt на renamed-text.txt

os.rename("text.txt", "renamed-text.txt")

Функция os.rename() принимает 2 аргумента: имя файла или папки, которые нужно переименовать и новое имя.

## **Перемещение файлов**

Функцию os.replace() можно использовать для перемещения файлов или каталогов:

# заменить (переместить) этот файл в другой каталог

os.replace("renamed-text.txt", "folder/renamed-text.txt")

Стоит обратить внимание, что это перезапишет путь, поэтому если в папке folder уже есть файл с таким же именем (renamed-text.txt), он будет перезаписан.

## **Список файлов и директорий**

# распечатать все файлы и папки в текущем каталоге

print("Все папки и файлы:", os.listdir())

Функция os.listdir() возвращает список, который содержит имена файлов в папке. Если в качестве аргумента не указывать ничего, вернется список файлов и папок текущего рабочего каталога:

Все папки и файлы: ['folder', 'handling-files', 'nested1', 'text.txt']

А что если нужно узнать состав и этих папок тоже? Для этого нужно использовать функцию os.walk():

# распечатать все файлы и папки рекурсивно

for dirpath, dirnames, filenames in os.walk("."):

# перебрать каталоги

for dirname in dirnames:

print("Каталог:", os.path.join(dirpath, dirname))

# перебрать файлы

for filename in filenames:

print("Файл:", os.path.join(dirpath, filename))

os.walk() — это генератор дерева каталогов. Он будет перебирать все переданные составляющие. Здесь в качестве аргумента передано значение «.», которое обозначает верхушку дерева:

Каталог: .\folder

Каталог: .\handling-files

Каталог: .\nested1

Файл: .\text.txt

Файл: .\handling-files\listing\_files.py

Файл: .\handling-files\README.md

Каталог: .\nested1\nested2

Каталог: .\nested1\nested2\nested3

Метод os.path.join() был использован для объединения текущего пути с именем файла/папки.

## **Удаление файлов**

Удалим созданный файл:

# удалить этот файл

os.remove("folder/renamed-text.txt")

os.remove() удалит файл с указанным именем (не каталог).

## **Удаление директорий**

С помощью функции os.rmdir() можно удалить указанную папку:

# удалить папку

os.rmdir("folder")

Для удаления каталогов рекурсивно необходимо использовать os.removedirs():

# удалить вложенные папки

os.removedirs("nested1/nested2/nested3")

Это удалит только пустые каталоги.

## **Получение информации о файлах**

Для получения информации о файле в ОС используется функция os.stat(), которая выполняет системный вызов stat() по выбранному пути:

open("text.txt", "w").write("Это текстовый файл")

# вывести некоторые данные о файле

print(os.stat("text.txt"))

Вывод:

os.stat\_result(st\_mode=33206, st\_ino=14355223812608232, st\_dev=1558443184, st\_nlink=1, st\_uid=0, st\_gid=0, st\_size=19, st\_atime=1575967618, st\_mtime=1575967618, st\_ctime=1575966941)

Это вернет кортеж с отдельными метриками. В их числе есть следующие:

* + st\_size — размер файла в байтах
  + st\_atime — время последнего доступа в секундах (временная метка)
  + st\_mtime — время последнего изменения
  + st\_ctime — в Windows это время создания файла, а в Linux — последнего изменения метаданных

Для получения конкретного атрибута нужно писать следующим образом:

# например, получить размер файла

print("Размер файла:", os.stat("text.txt").st\_size)

Вывод:

Размер файла: 19

На [этой странице](https://docs.python.org/3/library/os.html" \l "os.stat_result) больше об атрибутах.

## **Выводы**

Работать с файлами и каталогами в Python очень просто. Не имеет значения даже используемая операционная система, хотя отдельные уникальные для системы функции можно использовать: например, os.chown() или os.chmod() в Linux. Более подробно эта тема освещена в официальной документации Python.