



You have 1 free member-only story left this month. [Sign up for Medium and get an extra one](#)



Christopher Tao

Feb 7, 2021 · 8 min read

★ · Listen



Photo by [girlart39](#) on [Pixabay](#).

# 8 Must-Know File System Operations In Python

The essential for Python in tasks automation apps

In recent years, Python is known by a lot more people who are not programmers. This is not only because of its popularity in the area of Machine Learning but also because it could be used to automate a lot of repetitive works such as bulk editing files with certain patterns.

When the automation tasks are related to files, it is important to master the file system interfaces in Python. In this article, I'll introduce 8 file system operations in Python that are the most important and essential. Hope it can guide some learners to have an easier start.

All the source code are available in a Google Colab Notebook which has been made public. Please find the link in the last section and try them by yourself!

## 1. Show Current Directory

The first operation that I want to start with is to show the Current Working Directory (CWD). It is important because we might want to use relative paths in most of the time. Therefore, it is sometimes important to know where we are at the moment.

One of the reasons that the relative path is recommended because it doesn't create limitations of your application. No matter who copied your application, it should run on their computer as well.

OK. To show the current working directory, we need to use the OS library. This library is Python built-in, and most of the file operation functions can be found [here](#).

```
import os

os.getcwd()
```

It is easy to remember, so I would recommend to memorise it to save our time from Googling every time.

```
[1] import os

[2] print(f'Current Working Directory is: {os.getcwd()}')

Current Working Directory is: /content
```

Please be noticed that I’m using Google Colab. So, the path `/content` is actually an absolute path.

2. Check Directory or File Existing

Before introducing any functions for creating, we need to know that it is a good manner to check whether something is already existing before create it. It is not only because there might be an error, but also avoiding to overwrite something accidentally.

The function `os.path.exists()` takes a string type argument, which can be either the name of a directory or a file.

In my case, I’m using Google Colab and there is a folder automatically created called “sample\_data” every time a new notebook is provisioned. If I want to check whether there is such a directory, the following code will do.

```
os.path.exists('sample_data')
```

```
[3] os.path.exists('sample_data')

True
```

If we want to check the existence of a file, it also supports a file name as follows.

```
os.path.exists('sample_data/README.md')
```

```
[4] os.path.exists('sample_data/README.md')

True
```

Of course, if we check something not existing, it will return false.

```
[5] os.path.exists('foobar')

False
```

3. Join Path Components

In the previous example, I’ve deliberately shown a bad coding style. That is, in the string `'sample_data/README.md'` , I explicitly used the forward slash as the path component separator.

This is OK but not recommended. If you want your Python application to be able to run cross-platform, it is not safe to do so. For example, some older version Windows Operating System may only recognise backward slash `\` as separator.

Don’t worry, Python can handle this very well by using the function `os.path.join()` . Suppose we want to rewrite the function in the previous example using the join function, it should be as follows.

```
os.path.exists(os.path.join('sample_data', 'README.md'))
```

```
[7] os.path.exists(os.path.join('sample_data', 'README.md'))

True
```

### 4. Create a Directory

Now, let’s create a directory called `test_dir` in our working directory. We can simply use the function `os.mkdir()` .

```
os.mkdir('test_dir')
```

Let’s check whether it works.

```
[9] print(f"test_dir existing: {os.path.exists('test_dir')}")
    os.mkdir('test_dir')
    print(f"test_dir existing: {os.path.exists('test_dir')}")

test_dir existing: False
test_dir existing: True
```

However, it is important to know that creating an already existing directory will cause an exception.

```
[11] os.mkdir('test_dir')

-----
FileExistsError                                Traceback (most recent call last)
<ipython-input-11-48d7b58469aa> in <module>()
----> 1 os.mkdir('test_dir')

FileExistsError: [Errno 17] File exists: 'test_dir'
```

Therefore, it is recommended to always check the existence before creating it.

```
if not os.path.exists('test_dir'):
    os.mkdir('test_dir')
```

One more tip for creating directories. Sometimes, we may want to create sub-directories with 2 or more depth. If we still use `os.mkdir()` , we will need to do this multiple times. In this case, we can use `os.makedirs()` . This function will create all the intermediate directories just like `mkdir -p` flag in the Linux system.

```
os.makedirs(os.path.join('test_dir', 'level_1', 'level_2', 'level_3'))
```

The above code creates the directory tree as shown below.



### 5. Show Directory Content

Another useful function is `os.listdir()` that will show all the content of a directory. It is different from the `os.walk()` , where the latter will show everything “under” the directory recursively.

Most of the time, we might just want to loop the files in a directory, `os.listdir()` will be much easier to use, because it simply returns a list.

```
os.listdir('sample_data')
```



```
[13] os.listdir('sample_data')

['README.md',
 'anscombe.json',
 'california_housing_train.csv',
 'california_housing_test.csv',
 'mnist_train_small.csv',
 'mnist_test.csv']
```

However, sometimes we might want to do some advanced search. For example, we want to have all the CSV files in the “sample\_data” directory. In this case, the easiest way is to use `glob` library, which is also built-in to Python.

```
from glob import glob

list(glob(os.path.join('sample_data', '*.csv')))
```

```
[14] from glob import glob

[15] list(glob(os.path.join('sample_data', '*.csv')))
```

```
['sample_data/california_housing_train.csv',
 'sample_data/california_housing_test.csv',
 'sample_data/mnist_train_small.csv',
 'sample_data/mnist_test.csv']
```

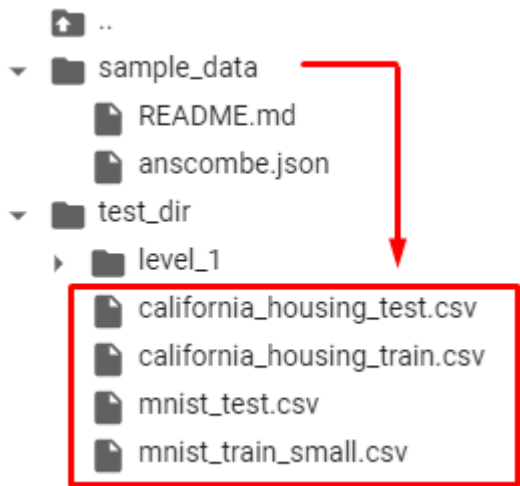
6. Move Files

Now, let’s move some files between different locations. The recommended way is to use the `shutil` library. Of course, it is Python built-in as well.

Suppose we want to move all the CSV files from the “sample\_data” directory to the “test\_dir” that we created in section 4. The code will be as follows.

```
import shutil

for file in list(glob(os.path.join('sample_data', '*.csv'))):
    shutil.move(file, 'test_dir')
```



There are definitely more different functions can achieve this. For example, we can still use the OS library if you don’t want to import extra libraries. Both the `os.rename` and `os.replace` will help us to move the files.

However, one of the drawbacks is that these two functions are not “smart” enough to let us move the files to a directory.

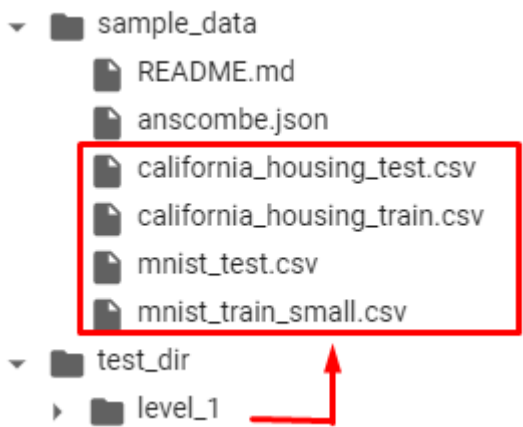
```
[21] for file in list(glob(os.path.join('test_dir', '*.csv'))):
      os.rename(file, 'sample_data')
```

```
-----
IsADirectoryError                                Traceback (most recent call last)
<ipython-input-21-9675c94bbbdd> in <module>()
      1 for file in list(glob(os.path.join('test_dir', '*.csv'))):
----> 2     os.rename(file, 'sample_data')
```

```
IsADirectoryError: [Errno 21] Is a directory: 'test_dir/california_housing_train.csv' -> 'sample_data'
```

To make it work, we have to explicitly specify the file name at the destination. The code below moves the CSV files back to the “sample\_data” directory.

```
for file in list(glob(os.path.join('test_dir', '*.csv'))):
    os.rename(
        file,
        os.path.join(
            'sample_data',
```



The function `os.path.basename()` here is to extract the file name from a path with any number of components.

The other function `os.replace()` will be exactly the same. However, the difference is that `os.replace()` is platform-independent, whereas `os.rename()` will only work on Unix/Linux system.

Another drawback is that these two functions from the OS library will not support to move files from different file systems, but `shutil` does.

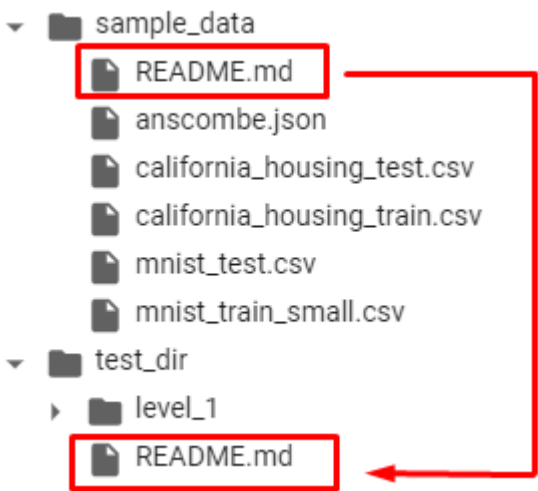
Therefore, it is highly recommended to use `shutil.move()` to move files in order to let your application behave consistently in different scenarios.

7. Copy Files

Similarly, `shutil` is recommended when we need to copy files for similar reasons.

If we want to copy the `README.md` from the “sample\_data” directory to the “test\_dir” directory, the function `shutil.copy()` will help.

```
shutil.copy(
    os.path.join('sample_data', 'README.md'),
    os.path.join('test_dir')
)
```



A tip for this function is that it can rename the file after copying it. Simply pass the new name as the second argument.

```
shutil.copy(
    os.path.join('sample_data', 'README.md'),
    os.path.join('test_dir', 'README(1).md')
)
```



## 8. Delete Directories or Files

Last but not the least, we need to delete files or directories sometimes. The `os` library will be enough most of the time.

When we want to delete a file, `os.remove()` should be used.

```
os.remove(os.path.join('test_dir', 'README(1).md'))
```

When we want to delete a directory, `os.rmdir()` will do.

```
os.rmdir(os.path.join('test_dir', 'level_1', 'level_2', 'level_3'))
```



However, it can only delete an empty directory. In the above screenshot, only the `level_3` directory can be removed. What if we want to remove the directory `level_1` recursively? In this case, we need help from the `shutil` library again.

```
[28] os.rmdir(os.path.join('test_dir', 'level_1'))

-----
OSError                                Traceback (most recent call last)
<ipython-input-28-352a126b9aab> in <module>()
----> 1 os.rmdir(os.path.join('test_dir', 'level_1'))

OSError: [Errno 39] Directory not empty: 'test_dir/level_1'
```

The function `shutil.rmtree()` will do the trick.

```
shutil.rmtree(os.path.join('test_dir', 'level_1'))
```

Remember to use it with care, because it will eliminate everything in the directory recursively.

## Summary

Get started

Sign In

Q

Search



Christopher Tao

7.1K Followers

👁 2.5M+ Reads

👤 7k+ Followers

🏆 Top 50 Writer

🎓 PhD

💻 Data Engineer/Machine Learning

LinkedIn

<https://www.linkedin.com/in/christopher-tao-5717a274/>

Follow

### More from Medium

 Ahm... in Toward...

2 Caveats to Avoid When Using Python Assert...






Photo by [Tumis](#) on Pixabay

In this article, I have introduced 8 essential file system operations and how to achieve them using Python. They are considered to be essential because they are the “building blocks” for any Python automation tasks that deal with multiple files.

It is also important to be aware that some of the functions are platform dependant while some others are not. Writing code in a good manner is crucial to any reliable application.

All the source code are available in the public Google Colab Notebook, which is ready to run.

**FileSystemOperationBasics**  
Edit description  
colab.research.google.com

**Join Medium with my referral link — Christopher Tao**  
As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...  
medium.com

If you feel my articles are helpful, please consider joining Medium Membership to support me and thousands of other writers! (Click the link above)


420 | 5



### Sign up for The Variable



By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

Get this newsletter

-  Art... in Better P...

**All 33 Reserved Keywords in Python**  

-  Giorg... in Towar...

**Checking if an Object Has an Attribute in...**  

-  Est... in Better P...

**A Detailed Guide to User Registration,...**  
