How to list all files of a directory?

Ask Question

How can I list all files of a directory in python and add them to a list?

python directory

edited Feb 6 at 19:56

martineau
43.6k 6 62 95

asked Jul 8 '10 at 19:31

duhhunjonn
10.1k 9 19 15

13 Related to How to get a list of subdirectories – rds Jan 5 '12 at 9:32

`os.listdir()` will get you everything that's in a directory - files and directories.

If you want *just* files, you could either filter this down using `os.path`:

```
from os import listdir
from os.path import isfile, join
onlyfiles = [f for f in listdir(mypath) if isfile(join(mypath, f))]
```

or you could use `os.walk()` which will yield two lists for each directory it visits - splitting into files and dirs for you. If you only want the top directory you can just break the first time it yields

```
from os import walk

f = []
for (dirpath, dirnames, filenames) in walk(mypath):
    f.extend(filenames)
    break
```

And lastly, as that example shows, adding one list to another you can either use `.extend()` or

```
>>> q = [1, 2, 3]
>>> w = [4, 5, 6]
>>> q = q + w
>>> q
[1, 2, 3, 4, 5, 6]
```

Personally, I prefer `.extend()`

edited Nov 22 '15 at 6:56

Martin Thoma
17.1k 22 140 254

answered Jul 8 '10 at 21:01

pycruft
20.9k 1 10 9

3 Doesn't seem to work on Windows with unicode file names for some reason. – cdiggins Jun 14 '13 at 16:21

23 A bit simpler: `(_, _, filenames) = walk(mypath).next()` (if you are confident that the walk will return at least one value, which it should.) – misterbee Jul 14 '13 at 20:56

3 Slight modification to store full paths: for (dirpath, dirnames, filenames) in os.walk(mypath): checksum_files.extend(os.path.join(dirpath, filename) for filename in filenames) break – okigan Sep 23 '13 at 21:31

65 `f.extend(filenames)` is not actually equivalent to `f = f + filenames`. `extend` will modify `f` in-place, whereas adding creates a new list in a new memory location. This means `extend` is generally more efficient than `+`, but it can sometimes lead to confusion if multiple objects hold references to the list. Lastly, it's worth noting that `f += filenames` is equivalent to `f.extend(filenames)`, *not* `f = f + filenames`. – Benjamin Hodgson ♦ Oct 22 '13 at 8:55

12 @misterbee, your solution is the best, just one small improvement: `_, _, filenames = next(walk(mypath), (None, None, []))` – bgusach Mar 5 '15 at 7:36

I prefer using the `glob` module, as it does pattern matching and expansion.

```
import glob
print glob.glob("/home/adam/*.txt")
```

Will return a list with the queried files:

```
['/home/adam/file1.txt', '/home/adam/file2.txt', .... ]
```

edited Dec 28 '14 at 3:24 — Cristian Ciupitu — 10.7k 3 31 49

answered Jul 9 '10 at 18:13 — adamk — 19.4k 4 38 51

6 that's a shortcut for listdir+fnmatch docs.python.org/library/fnmatch.html#fnmatch.fnmatch – Stefano Jul 1 '11 at 13:03

12 This returns some truly horrible slash inconsistency with me. `['C:/Users/Me/Downloads/temporary\\icon.ico']` – Anti Earth Jan 3 '13 at 11:35

10 For me it doesn't add to inconsistency I feed it with. Correct slashes at the input result in correct slashes at the output. – Antony Hatchkins Apr 24 '13 at 13:47

3 Beware that this returns the full path. – JI Xiang May 17 '16 at 14:31

5 to clarify, this does *not* return the "full path"; it simply returns the expansion of the glob, whatever it may be. E.g., given `/home/user/foo/bar/hello.txt`, then, if running in directory `foo`, the


```
import os
os.listdir("somedirectory")
```

will return a list of all files and directories in "somedirectory".

edited Jul 13 '16 at 19:05 — csano — 9,248 1 19 38

answered Jul 8 '10 at 19:35 — sepp2k — 238k 28 524 557

3 This returns the relative path of the files, as compared with the full path returned by `glob.glob` – JI Xiang May 17 '16 at 14:32

2 @JIXiang: `os.listdir()` always returns *mere filenames* (not relative paths). What `glob.glob()` returns is driven by the path format of the input pattern. – mklement0 Nov 30 '16 at 18:14

A one-line solution to get **only list of files** (no subdirectories):

```
filenames = next(os.walk(path))[2]
```

or absolute pathnames:

```
paths = [os.path.join(path,fn) for fn in next(os.walk(path))[2]]
```

edited Jan 14 '15 at 18:25 — Al Lelopath — 2,147 4 36 60

answered Jan 18 '14 at 17:42 — Remi — 8,538 4 35 37

5 Only a one-liner if you've already `import os`. Seems less concise than `glob()` to me. – ArtOfWarfare Nov 28 '14 at 20:22

3 problem with glob is that a folder called 'something.something' would be returned by glob('/home/adam/*.*') – Remi Dec 1 '14 at 9:08

2 On OS X, there's something called a bundle. It's a directory which should generally be treated as a file (like a .tar). Would you want those treated as a file or a directory? Using `glob()` would treat it as a file. Your

method would treat it as a directory. – ArtOfWarfare Dec 1 '14 at 19:44

Getting Full File Paths From a Directory and All Its Subdirectories

```
import os

def get_filepaths(directory):
    """
    This function will generate the file names in a directory
    tree by walking the tree either top-down or bottom-up. For each
    directory in the tree rooted at directory top (including top itself),
    it yields a 3-tuple (dirpath, dirnames, filenames).
    """
    file_paths = []  # List which will store all of the full filepaths.

    # Walk the tree.
    for root, directories, files in os.walk(directory):
        for filename in files:
            # Join the two strings in order to form the full filepath.
            filepath = os.path.join(root, filename)
            file_paths.append(filepath)  # Add it to the list.

    return file_paths  # Self-explanatory.

# Run the above function and store its results in a variable.
full_file_paths = get_filepaths("/Users/johnny/Desktop/TEST")
```

- The path I provided in the above function contained 3 files— two of them in the root directory, and another in a subfolder called "SUBFOLDER." You can now do things like:
- `print full_file_paths` which will print the list:
  - `['/Users/johnny/Desktop/TEST/file1.txt',`
    `'/Users/johnny/Desktop/TEST/file2.txt',`
    `'/Users/johnny/Desktop/TEST/SUBFOLDER/file3.dat']`

If you'd like, you can open and read the contents, or focus only on files with the extension


```
  if f.endswith(".dat"):

    print f

/Users/johnny/Desktop/TEST/SUBFOLDER/file3.dat
```

edited Jan 31 '14 at 15:54 — WBAR — 1,827 2 18 52

answered Oct 11 '13 at 0:55 — Johnny — 1,074 7 8

I really liked adamk's answer, suggesting that you use `glob()`, from the module of the same name. This allows you to have pattern matching with `*` s.

But as other people pointed out in the comments, `glob()` can get tripped up over inconsistent slash directions. To help with that, I suggest you use the `join()` and `expanduser()` functions in the `os.path` module, and perhaps the `getcwd()` function in the `os` module, as well.

As examples:

```
from glob import glob

# Return everything under C:\Users\admin that contains a folder called wlp.
glob('C:\Users\admin\*\wlp')
```

The above is terrible - the path has been hardcoded and will only ever work on Windows between the drive name and the `\` s being hardcoded into the path.

```
from glob    import glob
from os.path import join

# Return everything under Users, admin, that contains a folder called wlp.
glob(join('Users', 'admin', '*', 'wlp'))
```

The above works better, but it relies on the folder name `Users` which is often found on Windows and not so often found on other OSs. It also relies on the user having a specific name, `admin`.

```
from glob    import glob
from os.path import expanduser, join

# Return everything under the user directory that contains a folder called wlp.
glob(join(expanduser('~'), '*', 'wlp'))
```

This works perfectly across all platforms.

Another great example that works perfectly across platforms and does something a bit different:

```
from glob    import glob
from os      import getcwd
from os.path import join

# Return everything under the current directory that contains a folder called wlp.
glob(join(getcwd(), '*', 'wlp'))
```

Hope these examples help you see the power of a few of the functions you can find in the standard Python library modules.

edited Oct 6 '14 at 17:36

answered Jul 9 '14 at 11:43

ArtOfWarfare
9,203 5 58 93

4 Extra glob fun: starting in Python 3.5, `**` works as long as you set `recursive = True`. See the docs here: docs.python.org/3.5/library/glob.html#glob.glob – ArtOfWarfare Jan 26 '15 at 3:24

1 this is awesome saved me from so much of hassle. – armak Sep 23 '16 at 9:33

```
def list_files(path):
    # returns a list of names (with extension, without full path) of all files
    # in folder path
    files = []
    for name in os.listdir(path):
```


edited Oct 7 '14 at 18:30

answered Jun 10 '14 at 16:16

Apogentus
1,856 14 21

2 how about pep8? – Yauhen Yakimovich Sep 26 '14 at 9:37

Since version 3.4 there are builtin *iterators* for this which are a lot more efficient than `os.listdir()`:

`pathlib`: *New in version 3.4.*

```
>>> import pathlib
>>> [p for p in pathlib.Path('.').iterdir() if p.is_file()]
```

According to PEP 428, the aim of the `pathlib` library is to provide a simple hierarchy of classes to handle filesystem paths and the common operations users do over them.

`os.scandir()`: *New in version 3.5.*

```
>>> import os
>>> [entry for entry in os.scandir('.') if entry.is_file()]
```

Note that `os.walk()` use `os.scandir()` instead of `os.listdir()` from version 3.5 and it's speed got increased by 2-20 times according to PEP 471.

Let me also recommend reading ShadowRanger's comment below.

edited Feb 3 at 18:08

answered Jun 18 '15 at 20:58

SzieberthAdam
1,374 7 21

Thanks! I think it is the only solution not returning directly a `list`. Could use `p.name` instead of the first `p` alternatively if preferred. – JeromeJ Jun 22 '15 at 12:36

1 Welcome! I would prefer generating `pathlib.Path()` instances since they have many useful methods I would not want to waste waste. You can also call `str(p)` on them for path names. – SzieberthAdam Jul 13 '15 at 14:56

2 Note: The `os.scandir` solution is going to be more efficient than `os.listdir` with an `os.path.is_file` check or the like, even if you need a `list` (so you don't benefit from lazy iteration), because `os.scandir` uses OS provided APIs that give you the `is_file` information for free as it iterates, no per-file round trip to the disk to `stat` them at all (on Windows, the `DirEntry` s get you complete `stat` info for free, on *NIX systems it needs to `stat` for info beyond `is_file` , `is_dir` , etc., but `DirEntry` caches on first `stat` for convenience). – ShadowRanger Nov 20 '15 at 22:38

I've found this to be the most helpful solution (using `pathlib`). I can easily get specific extension types and absolute paths. Thank you! – HEADLESS_0NE Mar 17 '16 at 15:33

You should use `os` module for listing directory content. `os.listdir(".")` returns all the contents of the directory. We iterate over the result and append to the list.

```
import os

content_list = []

for content in os.listdir("."): # "." means current directory
    content_list.append(content)

print content_list
```

answered Mar 23 '16 at 10:09

Harun Ergül
1,492 1 16 28

14 `content_list = os.listdir(".")` also works as it returns a list. – ExceptionSlayer Apr 16 '16 at 1:13

1 This also includes the directories, right? Not just the files? – Samuel Edwin Ward Oct 28 '16 at 17:03

You are right @SamuelEdwinWard . – Harun Ergül Oct 30 '16 at 9:14

path.

answered Jul 7 '15 at 10:12

Rajat Garg
413 1 5 17

If you are looking for python implementation of **find**, this is a recipe I use rather frequently:

```
from findtools.find_files import (find_files, Match)

# Recursively find all *.sh files in **/usr/bin**
sh_files_pattern = Match(filetype='f', name='*.sh')
found_files = find_files(path='/usr/bin', match=sh_files_pattern)

for found_file in found_files:
    print found_file
```

so I made a PyPI package out of it and there is also a github repository. I hope that someone finds it potentially useful for his code.

edited Mar 17 '16 at 14:38

answered Apr 10 '14 at 14:09

Yauhen Yakimovich
6,308 3 39 55

Returning a list of absolute filepaths, does not recurse into subdirectories

```
L = [os.path.join(os.getcwd(),f) for f in os.listdir('.') if
os.path.isfile(os.path.join(os.getcwd(),f))]
```

edited Dec 28 '14 at 3:27

Cristian Ciupitu
10.7k 3 31 49

answered Jun 13 '14 at 16:26

The2ndSon
268 2 7

1 maybe bit longer but v clear what it is doing – javadba Jun 8 '15 at 0:28

Python 3.5 introduced new, faster method for walking through the directory - `os.scandir()` .

Example:

```
for file in os.scandir('/usr/bin'):
    line = ''
    if file.is_file():
        line += 'f'
    elif file.is_dir():
        line += 'd'
    elif file.is_symlink():
        line += 'l'
    line += '\t'
    print("{}{}".format(line, file.name))
```

answered Jan 17 '16 at 18:17

enedil
632 3 10 26

List all files in a directory:

```
import os
from os import path

files = [x for x in os.listdir(directory_path) if
path.isfile(directory_path+os.sep+x)]
```

Here, you get list of all files in a directory.

edited Sep 14 '15 at 13:03

worenga
3,615 1 16 34

answered Aug 29 '15 at 17:44

shiminsh
864 9 10


```
>>> import os
>>> for file in os.listdir():
...     print(file)
```

to include them in a list, just do this list comprehension:

> x = [f for f in os.listdir()]

If you want to have the list of a particular directory, not the directory where the terminal (or cmd) is:

```
>>> files = os.listdir('G:/python')
>>> for file in files:
        print(file)
>>> # this will create a new label for the list of files
>>> x = [f for f in os.listdir('G:/python')]
```

In case you want to make a list of just a type of file (like .jpg, for ex.):

```
>>> x = [f for f in os.listdir('G:/pyhton') if f.endswith('.jpg')]
['hello.jpg','cat.jpg']
```

Python 2 works differently...

```
>>> import os
>>> for f in os.listdir(os.getcwd):
...     print f

>>> # create a list
>>> x = [f for f in os.listdir(os.getcwd())] # the current dir
```

The code above will print all the files in the current directory The same here:

```
>>> import os
>>> for f in os.listdir('.'):
...     print f
```

```
>>> # create a list
>>> x = [f for f in os.listdir('.')] # the current dir
```

To go up in the directory tree, you got to code like this:

```
>>> for f in os.listdir(..):
...     print f


>>> # create a list
>>> x = [f for f in os.listdir('..')] # the precedent dir
```

and this...

```
>>> for f in os.listdir('/'):
...    print f

>>> # create a list
>>> x = [f for f in os.listdir('/')] # the root dir
```

... this will print all the files in the root directory

```
>>> x = [f for f in os.listdir('F:/python')] # a dir...
```

edited Feb 1 at 19:11

answered Jan 3 at 15:36

Giovanni Gianni
148 1 8

1 You should include the path argument to listdir. – Alejandro Sazo Jan 3 at 15:47

I thought this for the case you open the cmd or the terminal from the directory that you want to 'explore' ... but I will add what you say.. – Giovanni Gianni Jan 3 at 16:01

1 It's definitely encouraged to include some context/explanation for code as that makes the answer more useful. – EJoshuaS Jan 3 at 16:07

1 I agree, but I did not notice something also, that python2 requires the argument whilst python3 is optional, If you improve the answer for both python versions would be great :) – Alejandro Sazo Jan 3 at 16:44


```
# -** coding: utf-8 -*-
import os
import traceback

print '\n\n'

def start():
    address = "/home/ubuntu/Desktop"
    try:
        Folders = []
        Id = 1
        for item in os.listdir(address):
            endaddress = address + "/" + item
            Folders.append({'Id': Id, 'TopId': 0, 'Name': item, 'Address':
endaddress })
            Id += 1

            state = 0
            for item2 in os.listdir(endaddress):
                state = 1
            if state == 1:
                Id = FolderToList(endaddress, Id, Id - 1, Folders)
        return Folders
    except:
        print "_____________________________ ERROR _____________________________\n" +
traceback.format_exc()

def FolderToList(address, Id, TopId, Folders):
    for item in os.listdir(address):
        endaddress = address + "/" + item
        Folders.append({'Id': Id, 'TopId': TopId, 'Name': item, 'Address':
endaddress })
        Id += 1

        state = 0
        for item in os.listdir(endaddress):
            state = 1
        if state == 1:
            Id = FolderToList(endaddress, Id, Id - 1, Folders)
    return Id

print start()
```

edited Dec 28 '14 at 3:25

answered Mar 7 '14 at 10:28

Cristian Ciupitu

barisim.net

 **10.7k** 3 31 49  **81** 1 3

If you care about performance, try `scandir`, for Python 2.x, you may need to install it manually. Examples:

```
# python 2.x
import scandir
import sys

de = scandir.scandir(sys.argv[1])
while 1:
    try:
        d = de.next()
        print d.path
    except StopIteration as _:
        break
```

This save a lot of time when you need to scan a huge directory, you do not need to buffer a huge list, just fetch one by one. And also you can do it recursively:

```
def scan_path(path):
    de = scandir.scandir(path)
    while 1:
        try:
            e = de.next()
            if e.is_dir():
                scan_path(e.path)
            else:
                print e.path
        except StopIteration as _:
                break
```

answered Mar 12 '16 at 9:31

 coanor
1,314 1 17 33


```
import os
def createList(foldername, fulldir = True, suffix=".jpg"):
    file_list_tmp = os.listdir(foldername)
    #print len(file_list_tmp)
    file_list = []
    if fulldir:
        for item in file_list_tmp:
            if item.endswith(suffix):
                file_list.append(os.path.join(foldername, item))
    else:
        for item in file_list_tmp:
            if item.endswith(suffix):
                file_list.append(item)
    return file_list
```

answered Nov 11 '16 at 12:48

 neouyghur
368 3 14

```
import dircache
list = dircache.listdir(pathname)
i = 0
check = len(list[0])
temp = []
count = len(list)
while count != 0:
  if len(list[i]) != check:
     temp.append(list[i-1])
     check = len(list[i])
  else:
    i = i + 1
    count = count - 1

print temp
```

answered Jul 25 '12 at 10:25

 shaji
93 3

14 dirchache is "Deprecated since version 2.6: The dircache module has been removed in Python 3.0." – Daniel Reis Aug 17 '13 at 13:58

By using `os` library.

```
import os
for root, dirs,files in os.walk("your dir path", topdown=True):
    for name in files:
        print(os.path.join(root, name))
```

answered Oct 15 '16 at 16:29

Sankar Raj
2,891 1 7 26

Using generators

```
import os
def get_files(path):
     for (dirpath, _, filenames) in os.walk('.'):
         for filename in filenames:
             yield os.path.join(dirpath, filename)
list_files = get_files('.')
for filename in list_files:
    print(filename)
```

answered Dec 2 '16 at 7:01

shantanoo
1,460 11 24

protected by matt Dec 18 '14 at 2:54

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).