Branch: master ▾ **hackerU_python** / lessonPlan / **lesson4_osModule.md** Find file Copy path

t-0-m-1-3 added folders 190da8f on Jul 30, 2018

1 contributor

361 lines (270 sloc) 9.65 KB Raw Blame History

The OS Module

The Operating System Module allows for manipulating objects related to the oeprating system across all OS. You can write OS independent functions that won't have to remember which way the `\` or `/` is supposed to go on a Windows/MAC/UNIX system

To use the module, we need to import it into our programs first. Import a module into out programs gives us the ability to utilize all the code that the module has inside of it in out own code.

Importing can also come in handy when you have code in other `.py` files you want to run inside of your own code. Much like how javascript has the `require()`

you can import some or all of a files contents and use them inside of your python program.

```
import os

os.rename("new.txt", "newer.txt")
```

The above example on how to rename files gives you a glimpse into the power behind the `os` module. Why not take a quick stroll down the `help('os')` page to see just how powerful the methods and classes inside of it can be

```
#!/usr/bin/python

import os
import sys

fileStat = os.stat(sys.argv[1])

if fileStat:
    print 'Filename: %s' % sys.argv[1]
    print 'Size in bytes: %d' %fileState.st_size
```

The snippet above will demonstrate how to use the `.st_size` method

Using walk()

`os.walk()` is a function that allows you to recursively scan a directory and obtain tuple containing tuples of information

```
import os
import sys
```

```
for (path, dirs, files) is os.walk(sys.argv[1]):
    print path
    print dirs
    print files
```

the function `sys.platform` will output the **OS** you are using as well!

Exercise

1. Write a script that tells the user if they are using Windows or Linux
2. Use Google to search for the *SYS* library
3. Read into a file all /var/log/messages and save only the log files that contain USB
4. Write a script that takes in file names and returns the byte size.

OS Library

| library function | Description | | | | | `os.chdir(path)` | change your current path | | `os.mkdir(newName)` | create a new directory | | `os.rmdir(DeleteDirName)` | delete a directory | | | |

In Some cases you need to provide the *explicit* path that you want to use

Exercise Determine the output

1. `os.path.curdir`
2. `os.path.isdir(dir)`
3. `os.path.isfile(file)`
4. `os.path.exists(dir)`
5. `os.path.getsize(filename)`

Exercise Check out the `signal` module and find some ways to use it.

Directory Details:

- Methods for traversing directories
- Listing files and their information
- Creating and Deleting directories and files
- test to check if something is a file or a directory.

Code:

```
import os
os.getcwd()  #get your path
os.mkdir("NewDirectory") #Create a new directory
os.listdir(".") #list all the files in a directory, needs a path parameter
```

Exercise

1. How to remove a directory?
2. Write a script that will list file and directories and say what they are

```
for item in os.listdir(".")
    if os.path.isfile(item):
        print item + " is a file"
    elif os.path.isdir(item):
        print item + " is a directory"
    else:
        print "unkown!"
```

os.stat():

Gives informatino about the files and directories and performs statistics on system calls, given a path.

| method| Description| | | | | `st_mode` | protection bits | | `st_ino` | inode number | | `st_dev` | device inode resides on | | `st_nlink` | number of hard links | | `st_uid` | owner's user id | | `st_gid` | owner's group id | | `st_size` | size of the file in bytes | | `st_atime` | last access time | | `st_mtime` | last time modified | | `st_ctime` | last metadata change | | `os.stat('dir1')` | gives stats about the directory `dir1` | | | |

os.walk():

Walk generates a list of file names an allows you to traverse a directory top-down or bottom-up. walk will yield a 3-tuple (dirpath, dirnames, filenames)

| method| Description| | | | | `os.walk('dir1')` | save the path, directory names, and file names in `dir1` | | | |

```
variables = os.walk('dir1') # assign os.walk() to a var
for i in variable:  # start an interation loop for the output the variable
    print i # print it
```

you can also take advantage of the 3 items returns by `os.walk()` and separate them into 3 different objects.

```
for dirpath, dirnames, filenames in os.walk('dir1'):
    print 'Current path: ', dirpath
    print 'Directories: ' , dirnames
    print 'Files: ', filenames
```

os.environ():

This is used to print out environment variables.

`os.environ.get("HOME")` will get the value for the environment variable `HOME`

os.access(path, mode):

use the real uid/gid to test for access to the path

`os.access(path, mode)`

| parameter | description | | | | | `path` | this is that path to be tested | | `mode` | should be `F_OK` to test for existence/access. | | `os.F_OK` | value pass as the mode param to test if path exists | | `os.R_OK` | tests the readability of the file | | `os.W_OK` | tests the writability of the file | | `os.X_OK` | test to see if the path is executable | | | |

Exercise

Write a script that will show all the duplicate files in your WindowsOS and print the ones larger than 10MB

```
import os
import hashlib
import time
dict2={}
dict={}
f=open("Hashed.log","wb")
h-open("Duplicated.log","wb")
path=raw_input("please enter the full path to scan:")
for root, dirs, files in os.walk(path):
    os.chdir(root)
    try:
        fls=os.listdir(".")
        for i in fls:
            data=hashlib.md5(open(i,'rb').read()).hexdigest()
            if data in dict:
                h.write(data+" ")
                h.write(root|"\\"+i)
                h.write("\n")
                dict[data]+=1
                dict2[data]+=os.stat(i).st_size
            else:
                dict[data] = 1
                dict2[data] = 0
            f.write(data+" ")
            f.write(root+"\\"+i)
            f.write("\n")
            h.flush()
            h.flush()
    except:
        pass
    h.close()
    f.close()

    sm=0
    m=open("DupSize.lob","w")
    print "Duplicated files bigger than 10MB: "
    for key, value in dict2.iteritems():
        if value >0:
            m.write(key)
            m.write(str(flaot(value)/1000000))
            m.write("MB")
            m.write("\n")
            sm += float(value)/1000000
        if value > 10000000:
            print "[+] ", key, float(value)/1000000, "MB"
    print "============================================"
    print "=======My Duplicator========================"
    print "============================================"
```

- Add a delete option
- Show the location the duplicated files

Code:

```
import glob
for item in glob.glob(os.path.join(".","*.py")):
    print item
for item in glob.glob(os.path.join(".","*.py")):
    print item
```

Forking:

Forking is one of the most important topics in Linux/Unix. When a process forks it creates a copy of itself. A fork in multithreading environment means that, a thread of execution is duplicated, creating a child form the parent. Identical but distinct

Fork operation crates a separate address space for the Child. The child process has an exact copy of all the memory of the parent. Both execute independently

the System function call `fork()` creates a copy of a process. The copy runs as a child of the call. The child process gets the data and code from the parent but it's own PID from the operating system.

The child then runs as an independent instance of the parent process. Negative returns mean errors occurred trying to fork.

Take Aways:

- Cloning a process, creates an identical process as the parent
- The execution thread is duplicated exactly at the point of the call to `fork()`
    - `0` in the child; `pid` in the parent
- **PID** is different for parent/child

When, How, and Why

- Dedicated task given from a parent to a child
- Parent and Child communicate using IPC
- Parent and Child binary remains the same

```
import os

def child_proicess():
    print "I'm the child process and my PID is: %d" %os.getpid()
    print "The child is exiting"

def parent_process():
    print "I'm the parent process with PID: %d" %os.getpid()
    childpid = os.fork()

    if cihldpod == 0:

        child_process()
    else:
        print "we are inside the parent"
        print "Our child has the PID: %d" %childpid
    while True:
        pass
parent_process()
```

The exit statement `os.exit(0)` in child function is necessary, if not the process would return into the parent process.

Spawning New Processes:

| `os.exec` functions| | | | | `os.execl` | | `os.execle` |

- Overlays the parents process with the child process.

```
import os

os.execvp("ping", ["ping", "127.0.0.1"])

# using ping, the first argument is the process name, and the second the address to ping
```

Threads In Python:

Simple thread using the thread module more complicated ones using the threading module.

```
import thread
import time

def worker_thread(id):
    count = 1
    while True:
        print "Thread with ID %d has counter value %d" %(id.count)
        time.sleep(2)
        count +=1
for i in rang(5):
    thread.start_new_thread(worker_thread, (i,))

    print "Main thread going to an infinite loop"

    while True:
        pass
```

`thread.start_neq_thread(function, args[, kwargs])`

Start a new thread and return its identifier. The thread execites and the funciton lists arguments (as a tuple) Optional `kwargs` argument specifies a dictionary of keyword arguments. When the function terminates with an unhandled exception it prints a stack trace. otherwise it continues.