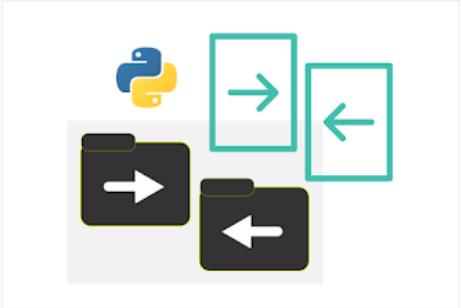
Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

viernes, 27 de mayo de 2016

Filecmp: comparando archivos y directorios



El módulo **filecmp** consta de varias funciones y una clase Python para comparar archivos y directorios.

Comparar dos archivos: cmp()

La función **filecmp.cmp()** compara dos archivos y devuelve **True** si son iguales o **False** en caso contrario.

filecmp.cmp(f1, f2, shallow=True)

El argumento **shallow** (superficial) con el valor predeterminado **True** se utiliza para que la función determine si dos archivos son iguales o diferentes comparando sólo la información de "estado" de los archivos, sin entrar a comparar su contenido. Para comparar también el contenido de los archivos es necesario establecer **shallow** con el valor **False**.

De toda la información de estado de un archivo, en una comparación superficial, sólo se utiliza el tipo de archivo, su tamaño y la fecha de modificación.

La información de estado de un archivo se puede obtener con la función **os.stat()** y, de ésta, se puede extraer con la función **filecmp._sig**() la necesaria para una comparación superficial.

Atención: en una comparación superficial de dos archivos pequeños (de unos pocos bytes) del mismo tipo, con diferente contenido pero con el mismo tamaño y creados en el mismo lapso de tiempo, el resultado de una comparación puede ser unas veces **True** y otras **False**. Esto sucede porque la diferencia de tiempo en el momento de la creación de los archivos puede llegar a ser tan ínfima que el sistema los puede llegar a representar con el mismo o similar valor. Lógicamente, si se crean dos archivos de varios miles de Kbytes del mismo tamaño, uno a continuación del otro, el valor devuelto en una comparación será siempre **True** porque la diferencia entre los tiempos, en este caso, sería distinguible por el sistema.

En cualquier caso, en una comparación superficial, no podemos extrañarnos si se dan por iguales dos archivos con distinto contenido o, por distintos, a dos archivos con igual contenido. Para una comparación superficial sólo importa la información de estado de los archivos comparados.

En el siguiente ejemplo se crean tres archivos en el directorio (donde se ejecute el código) y se realizan varias comparaciones:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#

import filecmp, shutil, os

factor = 1000000

# Crea un archivo con 3 x 1000000 bytes
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute

```
with open("arch1.txt", "w") as archivo1:
    archivo1.write("abc" * factor)

# Crea un segundo archivo con distinto contenido pero
# con igual tamaño (3 x 1000000 bytes)

with open("arch2.txt", "w") as archivo2:
    archivo2.write("123" * factor)

# Crea un tercer archivo copiando el primero. El archivo
# resultante tendrá la misma información de estado y el
# mismo contenido que el primero

shutil.copy2("arch1.txt", "arch3.txt")

# Por curiosidad, ahora se obtiene la información de estado
# que se utiliza en una comparación superficial

tup_est1 = filecmp._sig(os.stat("arch1.txt"))
tup_est2 = filecmp._sig(os.stat("arch2.txt"))
tup_est3 = filecmp._sig(os.stat("arch3.txt"))

# Se muestra la información de estado de los tres archivos

print("arch1,txt:", tup_est1)  # (32768, 3000000, 1464374862.0117595)
print("arch2.txt:", tup_est2)  # (32768, 3000000, 1464374862.0397596)
print("arch3.txt:", tup_est3)  # (32768, 3000000, 1464374862.0117595)

# A continuación, se realizan dos comparaciones superficiales
# y otras dos examinando el contenido; y se muestra el resultado

print(filecmp.cmp("arch1.txt", "arch2.txt", shallow=True))  # False
print(filecmp.cmp("arch1.txt", "arch3.txt", shallow=True))   # True
print(filecmp.cmp("arch1.txt", "arch2.txt", shallow=False))  # False
print(filecmp.cmp("arch1.txt", "arch3.txt", shallow=False))  # True

# Para trabajar con archivos pequeños y poder observar resultados
# "desconcertantes" asignar a la variable factor = 1 y ejecutar
# unas diez veces el código, observando sólo los resultados de la
# comparación. Es más que posible que el resultado de la primera
# comparación, a veces, sea True
```

Desde Python 3.4 con el método **filecmp.clear_cache()** es posible limpiar la caché que se utiliza en las operaciones de comparación. Esta limpieza será conveniente cuando los archivos sufran cambios muy frecuentes que afectan, como es normal, a la información del tiempo de modificación de los propios archivos.

Comparar lista de archivos en dos directorios: cmpfiles()

La función **filecmp.cmpfiles()** compara una lista de archivos en dos directorios y devuelve tres listas con:

- Los archivos encontrados en los dos directorios que son iguales;
- los archivos encontrados en los dos directorios que no son iguales;
- y los archivos no encontrados en alguno de los dos directorios.

filecmp.cmpfiles(dir1, dir2, common, shallow=True**)**

El argumento **common** es la lista de archivos a comparar.

El argumento **shallow** se utiliza de la misma manera que en la función **filecmp.cmp()**.

El siguiente ejemplo crea dos directorios con varios archivos. Después, se buscan y comparan en ellos los archivos de una lista común de archivos. Finalmente, se muestra el resultado alcanzado.

```
# Cada vez que se ejecute este código se crean dos
# directorios y se copian a ellos archivos del
# ejemplo anterior. (En caso de que los directorios
# existan de una ejecución anterior se borrarán,
# al principio, con todo su contenido).

if os.path.exists("dir1"): shutil.rmtree("dir1")
if os.path.exists("dir2"): shutil.rmtree("dir2")
os.mkdir("dir1")
os.mkdir("dir2")
```

simultáneamente varias operaciones en el mismo espacio de proceso se...

```
# En el directorio 'dir1' se copian los tres
# archivos del ejemplo anterior.

# Sin embargo, en el directorio 'dir2' se copia
# el archivo 'arch1.txt' dos veces pero con
# distinto nombre.

shutil.copy2("arch1.txt", "dir1/arch1.txt")
shutil.copy2("arch2.txt", "dir1/arch2.txt")
shutil.copy2("arch3.txt", "dir1/arch3.txt")
shutil.copy2("arch1.txt", "dir2/arch1.txt")
shutil.copy2("arch1.txt", "dir2/arch2.txt")

# Se declara una lista con una descripción breve
# de la información que se va a obtener en la
# comparación.

tipos = ["iguales", "desiguales", "errores"]

# Se declara una lista de nombres de archivos a
# comparar en los dos directorios

archivos = ["arch1.txt", "arch2.txt", "arch3.txt", "arch4.txt"]

# Se realiza la búsqueda y comparación de los archivos
# de la lista anterior en los dos directorios

resultado = filecmp.cmpfiles("dir1", "dir2",
                             archivos, shallow=False)

# Se imprime el resultado alcanzado (las tres listas:
# 'iguales', 'desiguales' y 'errores').

for t, r in zip(tipos, resultado):
    print(t, ":", r)

'''
Resultado
---------

iguales : ['arch1.txt']
desiguales : ['arch2.txt']
errores : ['arch3.txt', 'arch4.txt']
'''

# Observación:
# En la lista de 'desiguales' aparece 'arch2.txt' porque
# en 'dir1' se copia el archivo original y en 'dir2'
# se copia 'arch1.txt' con el nombre 'arch2.txt'.
# Cuando se comparan ambos directorios se detecta que
# su contenido es diferente
```

La clase dircmp

La clase **dircmp** permite crear nuevos objetos de comparación para comparar dos directorios (a y b).

filecmp.dircmp(a, b, ignore=None, hide=None**)**

- El atributo **ignore** es una lista de nombres a ignorar. Por defecto tiene el valor de **filecmp.DEFAULT_IGNORES**.
- El atributo **hide** es una lista de nombres a ocultar. Por defecto tiene el valor del directorio actual "." (**os.curdir**) y del directorio padre ".." (**os.pardir**)

Las comparaciones superficiales de archivos con la clase **dircmp** se realizan como con la función **filecmp.cmp()**.

Métodos

- **report()**: Muestra por la salida estándar el resultado de la comparación entre dos directorios.
- **report_partial_closure():** Muestra comparación parcial entre directorios a y b y subdirectorios comunes inmediatos.
- **report_full_closure()**: Muestra comparación parcial entre directorios a y b; y subdirectorios comunes (modo recursivo).

Atributos de la clase dircmp

Se emplea para acceder a variada información que se genera cuando se comparan dos árboles de directorios:

- **left**: el directorio a
- **right**: el directorio b
- **left_list**: contenido del directorio a (filtrado por ignore y hide)
- **right_list**: contenido del directorio b (filtrado por ignore y hide)
- **common**: lista de archivos y directorios comunes
- **left_only**: lista de archivos y directorios que sólo aparecen en a
- **right_only**: lista archivos y directorios que sólo aparecen en b
- **common_dirs**: subdirectorios comunes
- **common_files**: archivos comunes
- **common_funny**: archivos y directorios comunes con errores
- **same_files**: archivos idénticos tanto en a y b utilizando el operador de comparación de clase.
- **diff_files**: archivos diferentes tanto en a y b utilizando el operador de comparación de clase.
- **funny_files**: archivos que están tanto en a y b pero que no pueden ser comparados.
- **subdirs**: diccionario de subdirectorios con objetos de comparación

Desde Python 3.4 el atributo **filecmp.DEFAULT_IGNORES** devuelve la lista de directorios ignorados por defecto en la clase **dircmp**.

A continuación, un ejemplo basado en los anteriores que utiliza la clase **dircmp** para comparar los directorios 'dir1' y 'dir2'; en el que se obtiene mucha información gracias a los métodos y atributos disponibles:

```
# Dentro de los directorios de los ejemplos anteriores se
# crea un directorio llamado 'dir3' y se copia a este un
# archivo diferente a cada uno de ellos:

os.mkdir("dir1/dir3")
os.mkdir("dir2/dir3")
shutil.copy2("arch1.txt", "dir1/dir3/arch1.txt")
shutil.copy2("arch2.txt", "dir2/dir3/arch2.txt")

# Se comparan los directorios 'dir1' y 'dir2'

resultado = filecmp.dircmp('dir1', 'dir2')

# Se muestran distintos resultados

resultado.report()
resultado.report_partial_closure()
resultado.report_full_closure()

'''
Resultado de report()
---------------------
diff dir1 dir2
Only in dir1 : ['arch3.txt']
Identical files : ['arch1.txt']
Differing files : ['arch2.txt']
Common subdirectories : ['dir3']

Resultado de report_partial_closure()
-------------------------------------
diff dir1 dir2
Only in dir1 : ['arch3.txt']
Identical files : ['arch1.txt']
Differing files : ['arch2.txt']
Common subdirectories : ['dir3']

diff dir1/dir3 dir2/dir3
Only in dir1/dir3 : ['arch1.txt']
Only in dir2/dir3 : ['arch2.txt']

Resultado de report_full_closure()
----------------------------------
diff dir1 dir2
Only in dir1 : ['arch3.txt']
Identical files : ['arch1.txt']
Differing files : ['arch2.txt']
Common subdirectories : ['dir3']
```

```
diff dir1/dir3 dir2/dir3
Only in dir1/dir3 : ['arch1.txt']
Only in dir2/dir3 : ['arch2.txt']
'''

# Para finalizar, se accede a los atributos de la clase dircmp

print(resultado.left)  # dir1
print(resultado.right)  # dir2
print(resultado.left_list) #'arch1.txt','arch2.txt','arch3.txt','dir3'
print(resultado.right_list)  # 'arch1.txt', 'arch2.txt', 'dir3']
print(resultado.common)  # 'dir3', 'arch1.txt', 'arch2.txt']
print(resultado.left_only)  # ['arch3.txt']
print(resultado.right_only)  # []
print(resultado.common_dirs)  # ['dir3']
print(resultado.common_files)  # ['arch1.txt', 'arch2.txt']
print(resultado.common_funny)  # []
print(resultado.same_files)  # ['arch1.txt']
print(resultado.diff_files)  # ['arch2.txt']
print(resultado.funny_files)  # []
print(resultado.subdirs) #{'dir3':filecmp.dircmp object at 0xb6f82f0c}
print(resultado.subdirs['dir3'].report())
'''
Salida
------
diff dir1/dir3 dir2/dir3
Only in dir1/dir3 : ['arch1.txt']
Only in dir2/dir3 : ['arch2.txt']
None
'''
```

Relacionado:

- Difflib: encontrando las diferencias
- Explorando directorios con os.listdir, os.walk y os.scandir
- Filtrando archivos y directorios con glob y fnmatch
- Copiar, mover y borrar archivos/directorios con shutil

Ir al índice del tutorial de Python

Publicado por Pherkad en 15:22

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua