Содержание

1 Введение

В рамках данной курсовой работы мной был разработан программный модуль на языке Python, предоставляющий возможности работы с файловой системой. Модуль включает в себя 4 утилиты:

- Копирование деревьев каталогов
- Сравнение деревьев каталогов
- Поиск в деревьях каталогов
- Обход каталогов

Этот модуль относится к области системных приложений – сценариев, работающих с файлами, программами и окружением программ в целом. Исторически такие программы писались на непереносимых и синтаксически неочевидных языках оболочек, таких как командные файлы DOS, csh и awk. Однако в этой области ярко проявляются лучшие свойства Python. Например, простота использования Python и обширное многообразие встроенных библиотек упрощают использование развитых системных инструментов, таких как потоки выполнения, сигналы, ветвление процессов, сокеты и аналогичные им; такие инструменты намного сложнее использовать в неясном синтаксисе языков оболочек и в многоэтапных циклах разработки на компилируемых языках. Поддержка в Python таких идей, как ясность программного кода и объектно-ориентированное программирование, способствует созданию таких инструментов оболочки, которые можно читать, сопровождать и повторно использовать.

При использовании Python нет необходимости начинать с нуля каждый новый сценарий. Более того, в Python не только есть все интерфейсы, необходимые для разработки системных инструментов, но он также обеспечивает переносимость сценариев. При использовании стандартной библиотеки Python большинство системных сценариев, написанных на языке Python, автоматически становятся переносимыми на все основные платформы. Например, сценарий для обработки каталогов, написанный под Windows, обычно может выполняться и под Linux безо всякой правки исходных текстов: достаточно просто скопировать сценарий. Для разработки сценариев, обеспечивающих такой уровень переносимости, необходимо прикладывать некоторые усилия, тем не менее при разумном использовании Python может стать единственным средством, которым необходимо владеть для создания системных сценариев.

2 Основная часть

2.1 Копирование деревьев каталогов

Копирование в Windows путем перетаскивания мышью обладает рядом недостатков: копирование прерывается, как только будет обнаружен первый испорченный файл или как только встречался файл со слишком длинным или необычным именем. Сценарий cpall.py, представленный ниже реализует один из возможных способов копирования. С его помощью можно управлять действиями, которые выполняются при обнаружении проблемных файлов, например, пропустить файл с помощью обработчика исключения. Кроме того,

этот инструмент работает и на других платформах с тем же интерфейсом и таким же результатом.

```
# Порядок использования: "python cpall.py dirFrom dirTo".
# Рекурсивно копирует дерево каталогов. Действует подобно команде Unix "cp -r
# dirFrom/* dirTo", предполагая, что оба аргумента dirFrom и dirTo являются
# именами каталогов.
# Был написан с целью обойти фатальные ошибки при копировании файлов
# перетаскиванием мышью в Windows (когда встреча первого же проблемного файла
# вызывает прекращение операции копирования) и обеспечить возможность реализации
# более специализированных операций копирования на языке Python.

import os, sys
maxfileload = 1000000
blksize = 1024 * 500

def copyfile(pathFrom, pathTo, maxfileload=maxfileload):
    """
    Копирует один файл из pathFrom в pathTo, байт в байт;
        использует двоичный режим для подавления операций
        кодирования/декодирования и преобразований символов конца строки
    """
    if os.path.getsize(pathFrom) <= maxfileload:
        bytesFrom = open(pathFrom, 'rb').read()  # маленький файл читать целиком
        open(pathTo, 'wb').write(bytesFrom)
    else:
        fileFrom = open(pathFrom, 'rb')   # большие файлы - по частям
        fileTo   = open(pathTo,   'wb')   # режим b для обоих файлов
        while True:
        # прочитать очередной блок
            bytesFrom = fileFrom.read(blksize)
            if not bytesFrom: break       # пустой после последнего блока
            fileTo.write(bytesFrom)

def copytree(dirFrom, dirTo, verbose=0):
    """
    Копирует содержимое dirFrom и вложенных подкаталогов в dirTo,
        возвращает счетчики (files, dirs);
        для представления имен каталогов, недекодируемых на других платформах,
        может потребоваться использовать переменные типа bytes;
        в Unix может потребоваться выполнять дополнительные проверки типов файлов,
        чтобы пропускать ссылки, файлы fifo и так далее.
    """
    fcount = dcount = 0
    for filename in os.listdir(dirFrom):            # для файлов/каталогов
        pathFrom = os.path.join(dirFrom, filename)
        pathTo   = os.path.join(dirTo,   filename)  # расширить оба пути
```

```
        if not os.path.isdir(pathFrom):             # скопировать простые файлы
            try:
                if verbose > 1: print('copying', pathFrom, 'to', pathTo)
                copyfile(pathFrom, pathTo)
                fcount += 1
            except:
                print('Error copying', pathFrom, 'to', pathTo, '--skipped')
                print(sys.exc_info()[0], sys.exc_info()[1])
        else:
            if verbose: print('copying dir', pathFrom, 'to', pathTo)
            try:
                os.mkdir(pathTo)                    # создать новый подкаталог
                below = copytree(pathFrom, pathTo)  # спуск в подкаталоги
                fcount += below[0]                  # увеличить счетчики
                dcount += below[1]                  # подкаталогов
                dcount += 1
            except:
                print('Error creating', pathTo, '--skipped')
                print(sys.exc_info()[0], sys.exc_info()[1])
    return (fcount, dcount)

def getargs():
    """
    Извлекает и проверяет аргументы с именами каталогов, по умолчанию
        возвращает None в случае ошибки
    """
    try:
        dirFrom, dirTo = sys.argv[1:]
    except:
        print('Usage error: cpall.py dirFrom dirTo')
    else:
        if not os.path.isdir(dirFrom):
            print('Error: dirFrom is not a directory')
        elif not os.path.exists(dirTo):
            os.mkdir(dirTo)
            print('Note: dirTo was created')
            return (dirFrom, dirTo)
        else:
            print('Warning: dirTo already exists')
            if hasattr(os.path, 'samefile'):
                same = os.path.samefile(dirFrom, dirTo)
            else:
                same = os.path.abspath(dirFrom) == os.path.abspath(dirTo)
            if same:
                print('Error: dirFrom same as dirTo')
            else:
                return (dirFrom, dirTo)
```

```
if __name__ == '__main__':
    import time
    dirstuple = getargs()
    if dirstuple:
        print('Copying...')
        start = time.clock()
        fcount, dcount = copytree(*dirstuple)
        print('Copied', fcount, 'files,', dcount, 'directories', end=' ')
        print('in', time.clock() - start, 'seconds')
```

В этом сценарии реализована собственная логика рекурсивного обхода дерева каталогов, в ходе которого запоминаются пути каталогов источника и приемника. На каждом уровне она копирует простые файлы, создает каталоги в целевом пути и производит рекурсивный спуск в подкаталоги с расширением путей «из» и «в» на один уровень. Эту задачу можно запрограммировать и другими способами (например, в процессе обхода можно изменять текущий рабочий каталог с помощью функции os.chdir или использовать решение на основе функции os.walk, замещая пути «из» и «в» по мере их обхода), но на практике вполне достаточно использовать прием расширения имен каталогов при спуске.

Стоит обратить внимание на повторно используемую в этом сценарии функцию copyfile – на тот случай, если потребуется копировать файлы размером в несколько гигабайтов, она, исходя из размера файла, решает, читать ли файл целиком или по частям (при вызове без аргументов метода read файла он загружает весь файл в строку, находящуюся в памяти). Я выбрал достаточно большие размеры для читаемых целиком файлов и для блоков, потому что чем больший объем будет читаться за один подход, тем быстрее будет работать сценарий. Это решение гораздо эффективнее, чем могло бы показаться на первый взгляд, – строки, остающиеся в памяти после последней операции чтения, будут утилизироваться сборщиком мусора, и освободившаяся память будет повторно использована последующими операциями. Здесь используется двоичный режим доступа к файлам, чтобы подавить кодирование/декодирование содержимого файлов и преобразование символов конца строки – в дереве каталогов могут находиться файлы самых разных типов.

Стоит заметить также, что сценарий при необходимости создает целевой каталог, и перед началом копирования предполагает, что он пуст, – следует удалить целевой каталог перед копированием нового дерева с тем же именем, иначе к дереву результата могут присоединиться старые файлы (мы могли бы автоматически очищать целевой каталог перед копированием, но это не всегда бывает желательно). Кроме того, данный сценарий пытается определить – не являются ли исходный и конечный каталоги одним и тем же каталогом. В Unix-подобных системах, где есть такие структуры, как ссылки, функция os.path.samefile проделывает более сложную работу, чем простое сравнение абсолютных имен файлов (разные имена файлов могут означать один и тот же файл). Ниже приводится пример копирования большого дерева примеров книги в Windows. При запуске сценарию необходимо указать имена исходного и целевого каталогов, перенаправить вывод сценария в файл, если возникает слишком много ошибок, чтобы можно было прочитать все сообщения о них сразу (например, > output.Копирование деревьев каталогов 421 txt), и при необходимости выполнить команду оболочки **rm -r** или **rmdir /S** (или аналогичную для соответствующей платформы), чтобы сначала удалить целевой каталог:

```
C:\...\PP4E\System\Filetools> rmdir /S copytemp
```

```
copytemp, Are you sure (Y/N)? y
C:\...\P4E\System\Filetools> cpall.py C:\temp\P3E\Examples copytemp
Note: dirTo was created
Copying...
Copied 1430 files, 185 directories in 10.4470980971 seconds
C:\...\P4E\System\Filetools> fc /B copytemp\P3E\Launcher.py
C:\temp\P3E\Examples\P3E\Launcher.py
Comparing files COPYTEMP\P3E\Launcher.py and C:\TEMP\P3E\EXAMPLES\P3E\
LAUNCHER.PY
FC: no differences encountered
```

Можно воспользоваться аргументом verbose функции копирования, чтобы проследить, как протекает процесс копирования. В этом примере за 10 секунд было скопировано дерево каталогов, содержащее 1430 файлов и 185 подкаталогов, – на довольно низкопроизводительном ноутбуке (для получения системного времени была использована встроенная функция time.clock). Каким же образом этот сценарий справляется с проблемными файлами следующим образом: он перехватывает и игнорирует исключения и продолжает обход. Чтобы ско- пировать все хорошие файлы, достаточно выполнить команду такого вида:

```
C:\...\P4E\System\Filetools> python cpall.py G:\Examples C:\P3E\Examples
```

Поскольку на моем компьютере, работающем под управлением Windows, привод CD доступен как диск «G:», эта команда эквивалентна копированию путем перетаскивания элемента, находящегося в папке верхнего уровня на компакт-диске, за исключением того, что сценарий Python восстанавливается после возникающих ошибок и копирует остальные файлы. В случае ошибки копирования он выводит сообщение в стандартный поток вывода и продолжает работу. При копировании большого количества файлов, вероятно, будет удобнее перенаправить стандартный вывод сценария в файл, чтобы позднее его можно было детально исследовать. Вообще говоря, сценарию cpall можно передать любой абсолютный путь к каталогу на компьютере, даже такой, который обозначает устройство, например привод CD. Для выполнения сценария в Linux можно обратиться к приводу CD, указав такой каталог, как */dev/cdrom*. После копирования дерева каталогов таким способом можно проверить получившийся результат. Чтобы увидеть, для этого мной была написана следующая утилита

2.2 Сравнение деревьев каталогов

3 Заключение

4 Список литературы

:

1. `https://tinyurl.com/y3aj6pku`
   - Linked Lists in Detail with Python Examples: Single Linked Lists
2. `https://docs.python.org/3.5/tutorial/index.html` - The Python Tutorial
3. `https://docs.python.org/3/library/pathlib.html` - pathlib documentation

4. https://docs.python.org/3/library/os.html#module-os - os documentation
5. https://docs.python.org/3/library/argparse.html - argparse documentation
6. https://docs.python.org/3/library/tarfile.html#module-tarfile - tarfile documentation