Даниил Шидловский

Фотоальбом про запас

Язык программирования Python благодаря ясному и продуманному синтаксису помогает легко и непринужденно решать проблемы системного администрирования и практические задачи, оставляя свободное время. Это позволяет быть в хорошем настроении (ведь все получается быстро) и уделять внимание окружающим.

Если программирование — ваш хлеб насущный, то скорость кодирования позитивно скажется на доходах. А в моем случае небольшой скрипт на Python значительно улучшил мою, так сказать, семейную жизнь. Позвольте рассказать об этом подробнее.

Ради девушки своей мечты, я, как и многие мужчины, был готов к проявлению самых лучших творческих способностей. Но я не умею ни петь, ни рисовать, ни играть на музыкальных инструментах. Только компьютер у меня всегда под рукой и Python, конечно. И это помогло! Я создал скрипт для резервирования фотоколлекции на Gmail. Так программист выиграл у диджея! Ура!

В данной статье мы не просто подробно разберем написание программы, но и рассмотрим весь процесс поэтапно — от постановки задачи до готового решения. Более того, после небольших изменений вы сможете использовать этот скрипт для создания резервных копий как своих, так и пользовательских данных. И пока компьютер работает, оглянитесь — вокруг много красивых девушек, ждущих вашего внимания. Важное замечание: не забудьте купить цветы!

Вводные данные

Самостоятельно сделанные фотографии — самое ценное, кроме сногсшибательной внешности, что есть у (теперь уже) моей девушки. По последним подсчетам, их около 1800 штук. Все снимки хранятся на винчестере, который может уничтожить «сокровища» в любой момент. Но это можно предотвратить.

Наша цель — написать такую программу, которая будет заниматься проверкой всех файлов в каталоге и отправлять новые или измененные файлы заданных типов по почте на Gmail. Плюс, конечно, неплохо было бы научиться чему-нибудь новому, а также получить компоненты (модули) для повторного использования. Вообще, один из самых эффективных способов реализовать задуманное, в том числе и в программировании, — четко представлять конечный результат и промежуточные шаги, ведущие к цели.

Для нашей программы итог представлен в виде четырех файлов. Непосредственно ее текст находится в файле jpg2gmail.py. Входные данные — рабочий каталог, настройки сети, почтовый адрес и что-нибудь еще, что может понадобиться, — в файле jpg2gmail.ini. В jpg2gmail.db мы станем хранить данные об отправленных файлах. Тогда jpg2gmail.log будет содержать журнал работы нашей программы.

Мы можем использовать наш скрипт в качестве импортируемого модуля таким образом:

```
>>> import jpg2gmail
>>> print jpg2gmail.test()      # выдаст количество файлов для отправки
>>> jpg2gmail.upload(5)         # отправит пять файлов
```

Или вот так из командной строки:

```
python jpg2gmail.py test
# распечатает количество файлов, готовых к отправке
python jpg2gmail.py upload 10
# отправит десять файлов
```

Предварительная подготовка

Для хранения параметров создадим файл jpg2gmail.ini следующего содержания:

```
[main]
# адрес почтового ящика хранилища
destination_address = your.name@gmail.com
# адрес, присвоенный провайдером
from_address = myname@provider.com
# SMTP-сервер провайдера
smtp_server = smtp.your.provider.com
# директория с нужными файлами
work_dir = /home/tanya/images
```

Теперь разобьем нашу задачу на несколько мелких и прямо из интерпретатора Python найдем лучший способ их решить. Во-первых, как нам получить вводные данные?

```
>>> from ConfigParser import ConfigParser
>>> config = ConfigParser()
>>> config.read("jpg2gmail.ini")
>>> destination_address = config.get('main' , 'destination_address')
```

А как обойти все файлы в каталоге?

```
>>> import os                    # импортируем стандартный модуль
>>> print os.walk.__doc__  # читаем документацию к нужному методу и ...
# ... узнаем, что, для того чтобы получить список всех файлов в директории, нужно сделать так:
>>> result = []
>>> for dirpath, dirnames, filenames in os.walk("/home/tanya/images"):
...                 for file_name in filenames:
...                         result.append(os.path.join(dirpath, file_name))
>>>
```

Где хранить сведения об отправленных файлах? Используем стандартный модуль anydbm:

```
<listing>
>>> help("anydbm")                    # узнаем, как пользоваться модулем
>>> import anydbm                     # импортируем его
>>> db = anydbm.open("jpg2gmail.db", "w")      # открываем файл данных
```

База данных db работает как обычный словарь (dict). В качестве ключа используем полный путь к файлу, значение — дата последней модификации. Это предполагает, что при отправке файла мы делаем следующее:

```
>>> db[file_path] = str(os.path.getmtime(file_path)
>>> # os.getmtime выдает дату последней модификации файла
```

Информация о факте отправки сохраняется в jpg2gmail.db. Тут необходимо остановиться и немного подумать. Может, вы найдете лучшее решение?

Определяем, нужно ли отправлять данный файл? Верное решение складывается из ответов на три других вопроса.

Заданного ли типа этот файл? Если нет, пропускаем его:

```
>>> import mimetypes
>>> mimetypes.guess_type(file_name)[0] == 'image/jpeg'
```

Был ли этот файл уже загружен?

```
>>> db.has_key(file_path)
```

Если да, проверим, изменялся ли он после последней загрузки?

```
>>> str(os.path.getmtime(file_path)) == db[file_path]
```

Давайте подумаем, как превратить файл в электронное сообщение с присоединенными данными? Для меня это был самый сложный вопрос, но, немного поискав в Сети, я обнаружил готовый пример и практически без изменений использовал его здесь. Не хочу повторяться, поэтому объяснения последуют позже, когда доберемся до класса MessageManager.

Как отправить сообщение по электронной почте? Убедитесь сами, что проще простого:

```
>>> import smtplib
>>> S = smtplib.SMTP(smtp_server)
>>> S.sendmail(from_address, [destination_address], "message as string")
```

Теперь группируем задачи и создаем сущности для их решения. То есть описываем классы, методы и функции. Для начала создадим файл jpg2gmail.py следующего содержания:

Листинг

```
00  # ! /usr/bin/python
01  # -*-coding:utf8-*-
02  # jpg2gmail.py
03  '''
04  Usage:
05  $ python jpg2gmail.py test
06  $ python jpg2gmail.py upload 10
07  or
08  >>> import jpg2gmail
09  >>> jgp2gmail.test()
10  >>> jpg2gmail.upload()
11  '''
12  import MimeWriter, base64, StringIO, smtplib, os,
    mimetypes, sys, anydbm, ConfigParser, datetime
```

Комментарии

00	Стандартное начало для скриптов на Python: указан путь к интерпретатору
01	Указаны кодировки, в которых набраны неанглийские символы
02	Название файла
03–11	Документация к модулю. Мы всегда можем к ней обратиться через переменную __doc__
12	Импортируем все нужные для работы модули

Класс, управляющий сообщениями, — MessageManager

На MessageManager в нашем случае возложены две задачи — создать текстовое сообщение, содержащее прикрепленный файл и пригодное для отправки через SMTP-сервер, и отослать его.

Листинг

```
15  class MessageManager:
16          '''Create message from file path and send them via smtp server '''
17  def __init__(self, smtp_server, from_address, to_address):
18  self.sender = smtplib.SMTP(smtp_server)
19  self.from_address = from_address
20  self.to_address = to_address
21
22  def make_message(self, file_path):
```

```
23                      file_name = file_path.split(os.path.sep)[-1]
24                      subject=file_path.split(os.path.sep)[-2]
25
26                      message = StringIO.StringIO()
27                      writer = MimeWriter.MimeWriter(message)
28
29
30                      writer.addheader('MIME-Version', '1.0')
31                      writer.addheader('Subject', subject)
32                      writer.addheader('From', self.from_address)
33                      writer.addheader('To', self.to_address)
34
35                      writer.startmultipartbody('mixed')
36
37                      part = writer.nextpart()
38                      body = part.startbody('text/plain')
39                      body.write('Created by  ... \n')
40
41                      part = writer.nextpart()
42                      part.addheader('Content-Transfer-Encoding',
        'base64')
43                      body=part.startbody('application/jpg;
        name=%s' % file_name)
44
45                      fp=open(file_path, 'rb')
46                      body.write(base64.encodestring(fp.read()))
47                      fp.close()
48
49                      writer.lastpart()
50                      return message
51
52          def send_message(self, message):
53              return self.sender.sendmail(self.from_address,
        [self.to_address], message.getvalue())
```

Комментарии

15	Начало описания класса
16	Строка документации. Получаем доступ так: myinstance.__doc__
17	Описываем метод __init__ — конструктор экземпляра класса. Этот метод вызывается только тогда, когда необходимо создать экземпляр класса
18	Указана сущность, ответственная за отправку. Мы будем использовать ее в строке 53
19, 20	Сохраняем данные для дальнейшего использования
22	Начало функции make_message. Обратите внимание на обязательное использование ключевого слова self
23	Получаем имя файла из полного пути к нему. Это последний элемент массива, который имеется после разбиения пути к файлу с помощью команды split os.path.sep выдает разделитель для текущей операционной системы (“/” — для Unix, “\\” — для Windows)
24	Тема почтового сообщения — имя папки: получаем его, разбив путь к файлу и выбрав предпоследний элемент
26–38	Используем готовое без лишних вопросов
39	Тело почтового сообщения заполняем по своему усмотрению
40-44	Здесь представлены строки, необходимые для создания сообщения
45-47	Открываем файл, кодируем его содержимое по стандарту base64, записываем в сообщение и закрываем документ
48–51	Без комментариев — все и так понятно
52	Отправляем сообщение. Message — экземпляр класса StringIO. Получаем из него строку методом getvalue()
53	Возвращает пустой словарь, если все прошло успешно. Если есть проблемы, словарь содержит данные об ошибках. Обрабатываем в строке 95

Класс Logger, отвечающий за ведение лога

Листинг

```
54  cclass Logger:
55                      def __init__(self, file_name):
56                          self.file_name=file_name
57                      def write_line(self, txt):
58                          fp = open(self.file_name, 'a')
59                          fp.write('%s %s \n' %
    datetime.datetime.now().strftime('%d/%m/%Y:%H:%M:%S'),
    txt))
60                          fp.close()
```

Комментарии

55-56	В конструкторе передаем имя файла для ведения журнала
58	Открываем файл для записи. Если бы вместо 'a' было 'w', все данные перезаписывались бы. А так добавляются строки
59	Записываем строку в журнал: datetime.datetime.now() выдает текущее время; ftime('%d/%m/%Y:%H:%M:%S') позволяет получить строку вида 4/10/2005:2:50:46, как в логах Apache
60	Закрываем файл

Класс-обработчик — Handler

Данный класс определяет, есть ли необходимость отсылать файлы из полученного списка, а затем отправляет выбранные и отмечает в базе факт их отправки.

Листинг

```
61  class Handler:
62
63      def __init__(self, message_manager, logger, db_file_name):
64          self.manager = message_manager
65          self.logger=logger
66          self.db_file_name = db_file_name
67
68
69      def get_db(self):
70      if os.path.isfile(self.db_file_name):
71                      return anydbm.open(self.db_file_name, 'w')
72      return anydbm.open(self.db_file_name,'c')
73
74      def isSuitable(self, file_path):
75                      if mimetypes.guess_type(file_path)[0] !=
    'image/jpeg':
76                  return 0
77
78              db = self.get_db()
79                      if db.has_key(file_path):
80                          if str(os.path.getmtime(file_path))==
    db[file_path]:
81                              db.close()
82                              return 0
83              db.close()
84              return 1
85
86      def mark_as_uploaded(self, file_path):
87          db = self.get_db()
88          db[file_path] = str(os.path.getmtime(file_path))
89          db.close()
```

```
90
91          def process(self, files, qty=None):
92              if qty:
93                  files = files[:qty]
94              for file_path in files:
95                  error_dict = self.manager.send_message(self.mana-
    ger.make_message(file_path))
96                  if error_dict:
97                      self.logger.write_line('can\'t send file %s, %s ' %
    (file_path, str(error_dict)))
98                  else:
99                      self.mark_as_uploaded(file_path)
100                     self.logger.write_line('%s was uploaded' %
    file_path)
```

Комментарии

61	Начало описания класса
63-66	Передаем конструктору экземпляр MessageManager; экземпляр Logger и имя файла — базы данных, в которых хранятся сведения об отправленных фотографиях
69	Метод для получения доступа к базе данных
70-71	Если файл существует, открываем его для записи
72	Если файла нет, создаем его. Но при этом в любом случае возвращаем экземпляр для работы с базой данных
74	Смотрим, нужный ли файл указан
75-76	Если это не JPG-файл, то возвращаем 0 — он нам не подходит. Соответственно, отправлять его не нужно
78	Получаем доступ к базе данных
79	Есть такой ключ? То есть, мы этот файл уже отправляли?
80	Если отправляли, надо выяснить, был ли изменен документ с момента последней отправки. (В качестве ключа здесь мы используем полный путь к файлу, а в качестве значения — дату последней модификации.) Если дата последней модификации совпадает с данными, сохраненными в базе данных, то отправлять этот файл не нужно — возвращаем 0
81, 83	Не забываем закрывать базу. В противном случае, если компьютер даст сбой, то уже отправленные файлы придется отсылать снова. Выполнив команду db.close(), мы сохраняем данные на винчестер
84	Во всех остальных случаях файл нужно отправлять. Теперь возвращаем 1
86	Отмечаем факт отправки файла
87/89	Открываем/закрываем базу
88	Заносим в базу информацию об отправленном файле. Ключ — путь, значение — дата последней модификации
91	Метод для отправки группы файлов: files — список с путями к файлам, которые нужно отправить; qty — необязательный параметр, указывающий количество файлов для отправки. Зачем это нужно? Все просто. Если, например, 1000 файлов готовы к отправке, и мы отошлем их все за один раз, то провайдер может воспринять такую активность с нашей стороны как рассылку спама, что приведет к блокировке почтового ящика. Поэтому лучше отправлять письма порционно
92	Параметр передан?
93	Обрезаем список файлов
94	Перебираем все файлы из списка
95	Даем команду экземпляру MessageManager создать сообщение из файла (make_message) и отправить его (send_message). Если все прошло успешно, error_dict окажется пустым
96-97	Если найдена ошибка, заносим данные в журнал
98-100	Ошибки нет — отмечаем факт успешной отправки

Класс-обходчик — Walker

Основная задача этого класса — просмотреть все файлы, находящиеся в данной директории (включая также и поддиректории), и запомнить пути к тем из них, которые позднее будут отправлены по электронной почте.

Листинг

```
101     class Walker:
102         def __init__(self, handler):
103             self.handler = handler
104             self.files_for_upload = []
105
106         def go(self, root):
107             for dirpath, dirnames, filenames in os.walk(root):
108                 for file_name in filenames:
109                     file_path = os.path.join(dirpath, file_name)
110                     if self.handler.isSuitable(file_path):
111                         self.files_for_upload.append(file_path)
```

Комментарии

101-103	Начинаем описывать класс. Конструктор получает экземпляр класса Handler в качестве аргумента
104	Создаем контейнер для хранения списка файлов, готовых к отправке
106	Указываем путь к директории, где хранятся нужные файлы
107-108	Обходим все файлы в директории
109	Вычисляем полный путь к файлу
110-111	Если файл подходящий, добавляем путь к нему в контейнер

Функция, которая проверяет аргументы, переданные в командной строке

Листинг

```
112     def test_arguments(log, argv):
113
114         def usage():
115                 print >>sys.stderr, 'don\'t understand argu-
    ments'
116                 print >>sys.stderr,__doc__
117                 sys.exit(2)
118
119         if len(argv)<2 or argv[1] not in ['test', 'upload']:
120                 usage()
121
122         if argv[1] == 'upload' and len(argv)==3:
123                 try:
124                     int(argv[2])
125                 except:
126                     usage()
```

Комментарии

112	Нужно проверить, правильно ли переданы аргументы в командной строке. Эта функция вызывается из main (мы рассмотрим ее далее)
114	Иногда полезно определить функцию в функции
115	Система отвечает на стандартный вывод фразой «don't understand arguments» — это означает, что она не понимает указанных аргументов. И …
116	… объяснит, как лучше поступить, напечатав документацию к модулю (см. комментарии к строкам 03–12)
117	Вывод сообщения ОС, что не все так хорошо, как нам хотелось бы

119-120	Если показано меньше двух аргументов (первый — это всегда путь к исполняемому в данный момент файлу), то это указывает на ошибку. Или если второй аргумент — не test или upload, то это тоже можно расценивать как ошибку
122-126	Если второй аргумент — upload, а третий невозможно преобразовать в целое число, то здесь тоже ошибка

| Функция main |

Управление программой происходит отсюда.

Листинг

```
127 def main(argv=None):
128   script_dir = os.path.dirname(sys.argv[0])
129   log = Logger(os.path.join(script_dir, 'jpg2gmail.log'))
130
131   if  argv is None:
132                 argv = sys.argv
133   test_arguments(log, argv)
134
135   try:
136                 config = ConfigParser.ConfigParser()
137                 config.read(os.path.join(script_dir,
      "jpg2gmail.ini"))
138                 destination_address = config.get('main' ,
      'destination_address')
139                 from_address = config.get('main',
      'from_address')
140                 smtp_server = config.get('main', 'smtp_server')
141                 work_dir = config.get('main', 'work_dir')
142
143                 message_manager =
144   MessageManager(smtp_server, from_address,
      destination_address)
145                 handler = Handler(message_manager, log,
146   os.path.join(script_dir, 'jpg2gmail.db'))
147                 walker = Walker(handler)
148                 walker.go(work_dir)
149           except:
150                 for msg in sys.exc_info():
151                       log.write_line(msg)
152                 sys.exit(2)
153
154           if argv[1] == 'test':
155                 print '%s files ready for upload' % len(walk-
      er.files_for_upload)
156
157   if argv[1] == 'upload':
158                 if len(argv)==3:
159                       handler.process(walker.files_for_upload,
      int(argv[2]))
160                 else:
161                       handler.process(walker.files_for_upload)
```

Комментарии

127	Главная функция — точка входа в программу
128	Вычисляем путь к директории, в которой хранится исполняемый файл. Именно в ней должны находиться файлы jpg2gmail.ini, jpg2gmail.db и jpg2gmail.log
129	Создаем экземпляр класса Logger. os.path.join используется для вычисления полного пути к файлу журнала
131	Если аргументы командной строки не переданы напрямую, то получаем их с помощью модуля sys
133	Проверяем правильность аргументов.
135	Выражение try (и except в 149) поможет найти ошибки
136–141	Получаем текущие параметры из файла jpg2gmail.ini
143–148	Создаем основных игроков и отправляем Walker в путь. В этой ситуации можно слегка почувствовать себя современным магом
149–152	Если произошли ошибки, заносим сведения о них в журнал и выходим, сообщив операционной системе, что у нас проблемы
154-155	Если получена команда test, распечатываем количество готовых к отправке файлов
157	Если получена команда upload …
158	… и аргументов передано три (например, python jpg2gmail.py upload 7)
159	…даем команду на отправку указанного выше количества файлов
160-161	Если количество файлов не задано, отсылаем все файлы, готовые к отправке

Дополнительные функции, которые вызываются при импорте модуля, выглядят вот так:

```
>>> import jpg2gmail
>>> jpg2gmail.test()              # распечатает количество
файлов, готовых к отправке
>>> jpg2gmail.upload(2)   # отправит два файла
>>> jpg2gmail.upload()  # отправит все файлы, готовые к отправке
```

Листинг

```
162 def test():
163   main([None, 'test'])
164
165 def upload(qty=None):
166       if qty is None:
167             main([None, 'upload'])
168   else:
169             main([None, 'upload', qty])
170
171 if __name__== '__main__':
172       sys.exit(main())
```

Комментарии

162	Эта функция дает команду вывести на печать все файлы, готовые к отправке
163	Вызываем функцию main, эмулируя список аргументов командной строки. Первый аргумент — путь к скрипту — в этом случае не используется
165	Функция дает команду на отправку файлов
166–169	Если передан необязательный аргумент qty (количество), переназначаем его функции main вместе с командой upload. Если qty нет, то просто выполняем команду upload
171	Условие ___name__=='__main__' выполняется, только если скрипт запускается из командной строки, в случае импорта другим модулем этого не происходит
172	Закрывает сессию интерпретатора и возвращает результат выполнения функции main. Если main вернет 2, то все узнают, что что-то пошло не так

Ну вот мы и добрались до конца. Поздравляю! Если вы разобрались в этом не совсем простом примере, то в следующем номере журнала вас будет ждать приз — небольшой программный продукт (естественно, открытый), благодаря которому от клиентов не будет отбоя. |