Сергей Супрунов

Рабочая лошадка администратора

Язык программирования Python в последнее время становится все более популярным. Он по умолчанию включается в большинство дистрибутивов Linux, его можно установить на FreeBSD (кстати говоря, Perl с некоторых пор также не является частью системы, так что и в этом плане он уравнен в правах с Python). Многие сторонние программы наряду с поддержкой Perl обеспечивают и работу с Python (например, модуль mod_python к Apache, язык PL/pySQL в PostgreSQL).

Поднимать вечный спор «Perl vs Python» не будем: с обеих сторон можно привести массу доводов как «за», так и «против». На наш взгляд, здесь решающим фактором при выборе будет, скажем, психологическая совместимость языка и программиста. Например, некоторым Python пришелся по душе из-за строгого синтаксиса и очень удобной концепции повторного использования кода.

В данной статье мы рассмотрим несколько примеров использования Python для решения задач администрирования систем Unix (все примеры оттачивались на FreeBSD, но и на Linux должны работать не хуже). Предполагается, что большинство пользователей хотя бы в общих чертах знакомы с синтаксисом Python. Если вы не относитесь к их числу, переходите к следующему разделу, в котором коротко описаны основные моменты, необходимые для чтения представленного в статье кода.

Несколько вводных замечаний

Напомним основные моменты синтаксиса языка Python. Прежде всего, блоки кода в нем выделяются не операторными скобками, как в C или Perl, а отступами. Каждая команда записывается на одной строке (в принципе синтаксис позволяет объединять несколько команд в одну строку или переносить команду на следующую; однако это ухудшает читаемость кода и настоятельно не рекомендуется). Все строки одного блока (цикла, подпрограммы, ветвления и т. д.) должны быть выровнены на одинаковое количество пробельных символов (пробелами или табуляцией). Например:

```
for file in dir:
    print file
    print '----------------'
print file
```

В данном фрагменте телом цикла for являются первые две строки print (заданы с отступом относительно оператора for). Последняя строка будет исполнена после выхода из цикла, поскольку она уже не имеет отступа.

Как вы, должно быть, заметили, команды не нужно завершать символом «;» (он должен указываться только при размещении нескольких команд в одной строке). Поскольку перевод строки рассматривается как конец команды, для ее переноса символ конца строки должен экранироваться:

```
print a, b, c, d, e,
    f, g, h
```

Строку внутри кавычек можно переносить без экранирования, поскольку в этом случае перевод строки считается принад-

лежащим строковой константе (обязательно убедитесь, что этот символ не исказит отображение строки).

В Python идея повторного использования кода заложена в основу синтаксиса. Если вы когда-либо писали какую-то функцию (или получили готовую), то ее очень легко подключить к любому другому сценарию с помощью оператора import:

```
import sys
from Tkinter import *
```

В первой строке мы подключаем модуль sys, включенный в стандартную библиотеку Python. Во второй импортируются все функции модуля Tkinter. Причем в качестве модуля может рассматриваться любой файл, содержащий код Python. Отличие этих способов заключается в том, что при «простом» импорте для всех функций подгружаемого модуля создается свое пространство имен, и при обращении к той или иной функции (или переменной) следует указывать имя модуля (см. примеры ниже). А конструкция «from <module_name> import <list_of_functions>» добавляет импортируемые функции в пространство имен выполняемого сценария, что позволяет обращаться к функции без указания имени модуля. Естественно, импортируемое таким образом имя не должно конфликтовать с уже имеющимися. Кроме того, во втором случае вместо символа «*» можно явно перечислить функции, которые должны быть доступны в разрабатываемом сценарии.

Комментарии, как и в Perl, начинаются с символа «#».

Теперь рассмотрим несколько примеров решения различных задач администрирования.

Автоматическое редактирование файлов

Исходные условия

Имеется древовидная структура каталогов, содержащая текстовые файлы. Для определенности будем считать, что такой структурой является дерево веб-сайта некоторой компании (простые HTML-файлы). Пусть по ряду причин компания сменила название с Indians на Cowboys, что отразилось и на доменном имени сайта — вместо www.indians.com будет www.cowboys.ru.

Задача

Реализовать автоматическую корректировку всех HTML-файлов, заменив все названия и доменные имена новыми значениями. Будем считать, что регистр символов везде соблюдается безукоризненно, то есть нигде в тексте не встречается написание типа InDiAnS.

Решение

Сначала приведем полностью итоговый сценарий, учитывая его небольшой размер. Необходимые пояснения размещены ниже.

```
#!/usr/local/bin/python
import os, sys
from glob import glob
exts  = ['html', 'htm']
old = ['Indians', 'indians.com']
new = ['Cowboys', 'cowboys.ru']
DEBUG = 1
def indir(curdir):
    global space
    dir = glob('*')
    for file in dir:
        ext = os.path.splitext(file)[1][1:]
        if os.path.isdir(file):
            if DEBUG: print '%s---> %s' % ('   ' * space, file)
            os.chdir(file)
            space += 1
            indir(file)
            space -= 1
            os.chdir('..')
            if DEBUG: print '%s<--- %s' % ('   ' * space, file)
        elif exts.__contains__(ext):
            try:
                text = orig = open(file, 'r').read()
                i = 0
                for word in old:
                    text = text.replace(old[i], new[i])
                    i += 1
                if orig != text:
                    open(file, 'w').write(text)
                    if DEBUG: print '%s+ %s...' % \
                      ('   ' * space, file)
                else:
                    if DEBUG: print '%s= %s...' % \
                      ('   ' * space, file)
            except:
                print 'I/O ERROR while %s processing' % file
def Usage():
    print 'Usage: fe.py startdir'
    print
space = 0
if len(sys.argv) != 2:
    Usage()
else:
    startdir = sys.argv[1]
    if DEBUG: print 'Start with %s' % startdir
    os.chdir(startdir)
    indir(startdir)
```

Самая первая строка указывает операционной системе (если речь идет о Unix) путь к командному интерпретатору Python. Приведенный путь характерен для FreeBSD, в Linux это обычно /usr/bin/python. В Windows она воспринимается как обычный комментарий.

Здесь нам понадобились модули os и sys, а также функция «glob» из одноименного модуля. Все модули входят в стандартную поставку Python, искать их нигде не нужно. Задаем список обрабатываемых расширений (переменная exts) и два списка заменяемых значений — в первом старые строки, которые должны быть заменены, во втором — новые (оба списка должны строго соответствовать друг другу).

Установка значения «1» переменной DEBUG позволит получать в окне терминала подробные сообщения о процессе исполнения сценария.

Далее следует определение функции «indir» (оператор def). К ней мы вернемся немного позже, пока лишь отметим, что к ней принадлежат все строки, имеющие отступ.

Функция «Usage» выводит на экран сообщение о синтаксисе запуска сценария.

Последние строки сценария выполняют следующие действия: переменная space инициализируется значением «0» (она будет нужна в функции «indir»); проверяется количество переданных сценарию аргументов, и если оно не равно двум (имя самого сценария и один параметр), то вызывается функция «Usage». То, что передано в командной строке, можно получить из специального списка argv модуля sys. Переменная sys.argv[0] хранит имя запущенного сценария, поэтому первый аргумент имеет индекс 1. Если длина списка sys.argv равна двум, считывается аргумент, переданный в командной строке (мы будем передавать в сценарий имя каталога, являющегося корнем обрабатываемого дерева). Далее выводится отладочное сообщение; работа операторов if и print, как нам кажется, понятна без комментариев. Разве что следует обратить внимание на использование знакомест при выводе текстовой строки: в самой строке место, куда должно быть выведено значение строковой переменной, отмечается конструкцией %s, при обработке вместо нее подставляется конкретное значение соответствующей переменной — одной из перечисленных после символа «%».

Функция «chdir» модуля os делает текущим указанный каталог, а последней строкой вызывается функция «indir», которая и будет выполнять основную работу.

В функции «indir» мы сначала объявляем глобальную переменную. Без такого объявления space будет считаться локальной и не примет во внимание значение, установленное вне функции. Назначение переменной space рассмотрим несколькими абзацами ниже.

Функция «glob» модуля glob (поскольку она была импортирована в пространство имен нашего сценария, то при обращении к ней имя модуля не указывается) возвращает список имен файлов и каталогов, найденных в текущей папке и соответствующих указанному шаблону. В данном случае мы используем шаблон «*», поскольку имена подкаталогов могут быть любыми. Обратите внимание, что в Unix DOT-файлы (имена которых начинаются с точки, включая каталоги . и ..) не соответствуют шаблону «*» (чтобы убедиться в этом, наберите «echo *» в командной строке), поэтому дополнительных мер по исключению зацикливания на родительском каталоге предпринимать не требуется.

Таким образом, переменная dir теперь содержит список всех файлов и каталогов, находящихся в текущей папке.

Цикл for в Python позволяет простым и естественным образом проходить по всем элементам списка. В нашем случае тело цикла будет выполнено для каждого имени файла или каталога из dir. Текущий элемент заносится в переменную file.

Следующая строка, призванная дать нам расширение обрабатываемого файла, выглядит не совсем понятно:

ext = os.path.splitext(file)[1][1:]

Здесь функция «splitext», находящаяся в подмодуле path модуля os, разбивает полное имя файла, переданное как параметр, на две части — все, что стоит до расширения (до последней точки), и само расширение. Значения возвращаются в виде список, в котором нас интересует второй элемент. Рассматривая функцию os.path.splitext(file) как переменную-список (ее результат и будет занесен во временную переменную), мы применяем к ней обычную процедуру извлечения элемента по индексу. Таким образом, конструкция os.path.splitext(file)[1] будет представлять собой внутреннюю переменную, содержащую расширение файла. Однако в эту строку попадает и ведущая точка, то есть ее содержимое на данном этапе выглядит как .html, и эту точку следует убрать.

В Python символьная строка может рассматриваться как список букв (выполните в интерфейсе интерпретатора команду for a in ‘qwerty’: print a, чтобы убедиться в этом). Для списков существует особая операция — срез. Например, чтобы получить список, содержащий элементы с третьего по пятый некоторого существующего списка, можно выполнить следующее:

Sublist = list[3:6]

Таким образом, указывается диапазон индексов, причем элемент, имеющий индекс, указанный вторым, в результирующий список не включается (то есть sublist будет содержать элементы 3, 4 и 5). Если опущен первый из индексов, подразумевается «с начала списка», если опущен второй — «до конца списка». Теперь становится понятной и последняя запись разбираемой строки: в переменную ext записывается не вся строка расширения, а подстрока, начиная со второго символа и до конца. Как видите, интерпретатор Python позволяет осуществлять довольно сложную обработку без явного указания переменных.

Вернемся к рассмотрению функции «indir». Если очередной элемент из переменной dir представляет собой имя каталога (проверяется функцией «isdir» модуля os.path), то выполняется переход в этот каталог, рекурсивный вызов «indir» и возвращение в родительский каталог после завершения обработки.

Следует чуть более подробно остановиться на записи «’ ‘ * space», которая является переменной для заполнения первого знакоместа в отладочных сообщениях. Оператор «*» для строк означает «повторить указанное количество раз». Глобальная переменная space как раз эти «разы» и содержит, увеличиваясь при входе в подкаталог и уменьшаясь при возврате из него на один. В результате вывод на экран представлен «лесенкой», отражающей вложенность каталогов.

Если очередной элемент является регулярным файлом (конечно, есть еще и символьные ссылки, файлы-«дырки» различных устройств и т. д., но в данном примере мы про это «забудем»), то проверяем его расширение (часть elif оператора if). Здесь удобно использовать внутреннюю функцию «__contains__», которая применена к списку и возвращает «истину», если список содержит элемент, равный ее аргументу. Таким образом, дальнейшей обработке будут подвергаться только файлы, расширения которых упоминаются в списке exts, заданном в начале сценария.

Собственно обработка файла достаточно проста: его содержимое считывается в переменную text (копия — в переменную orig; Python допускает множественное присваивание), к которой в цикле применяется метод replace для каждой пары значе-

ний. Запись в файл выполняется только в том случае, если измененный текст (text) отличается от orig. Но поскольку операции с файлами небезопасны (может оказаться недостаточно прав на чтение или запись), их целесообразно выполнять внутри оператора «try .. except ..»: если при выполнении команд блока try возникнет ошибка, управление будет передано на блок except, и выполнение сценария продолжится.

Обратите также внимание, что при работе с файлом нигде не указывается его десткриптор. То есть мы используем тот же прием, что и ранее при отделении расширения от имени файла — неявные переменные. В классическом варианте чтение из файла выглядело бы следующим образом:

```
fd = open(file, 'r')
text = fd.read()
fd.close()
```

Кстати, поскольку неявная переменная-дескриптор существует только внутри данной команды, то при переходе на следующую файл будет автоматически закрыт, поэтому функцию «close()» вызывать не обязательно, чем мы и воспользовались.

На выводе рассмотренного сценария символом «+» будут отмечены файлы, в которых были сделаны корректировки, «=» — которые оставлены без изменений. Скорость работы оказалась достаточно приличной — в дереве общим объемом около 5 Мбайт (235 файлов, подлежащих обработке) тройная замена была выполнена за пару секунд.

Анализ лог-файлов Apache

Исходные условия

Имеется веб-сервер Apache, ведущий лог-файл (для определенности — /var/log/httpd-access.log) посещения страниц вида:

```
x.x.x.x — - [16/Jun/2005:07:06:44 +0400] "GET /city/city.htm
HTTP/1.0" 304 — "-" "ConveraCrawler/0.8
(+http://www.authoritativeweb.com/crawl)"
y.y.y.y — - [16/Jun/2005:07:33:11 +0400] "GET /map.html
HTTP/1.0" 200 3960 "-" "Googlebot/2.1
(+http://www.google.com/bot.html)"
```

Задача

Получить статистику посещений каждой из страниц, а также рассчитать общий объем отданных клиентам страниц.

Решение

Первая строка приведенного выше примера лог-файла соответствует ответу с кодом 304 (Not Modified) и интереса для нас не представляет (если, конечно, не потребуется собирать статистику и по таким ответам). В данном примере учитывать следует только ответы с кодом 200 (страница отдана клиенту).

Код сценария в данном случае заметно проще, но тоже имеет ряд особенностей:

```
#!/usr/local/bin/python
# -*- coding: koi8_r -*-
fd = open('/var/log/httpd-access.log', 'r')
stat = {}
cnts = {}
line = 'not empty'
while 1:
    line = fd.readline()
    if not line:
        break
    fields = line.split(' ')
    if fields[5] == '"GET' and fields[8] == '200':
        url = fields[6]
        if url.find('?') != -1:
            url = url[:url.index('?')]
        traf = int(fields[9])
    else:
        continue
    if stat.has_key(url):
        stat[url] += traf
        cnts[url] += 1
    else:
        stat[url] = traf
        cnts[url] = 1
fd.close()
statkeys = stat.keys()
statkeys.sort()
totaltrf = 0
totalcnt = 0
print '-' * 60
print '%-25s %12s %12s' % ('Страница', 'Обращений', 'Трафик')
print '-' * 60
for key in statkeys:
    print '%-25s %12d %12d' % (key, cnts[key], stat[key])
    totaltrf += stat[key]
    totalcnt += cnts[key]
print '-' * 60
print '%-25s %12d %12d' % ('Total:', totalcnt, totaltrf)
print '-' * 60
```

Вторая строка сценария указывает используемую кодировку, что необходимо, так как в данном примере используются символы, выходящие за рамки стандарта ASCII-128. Если кодировку не указать, то сценарий станет работать, но каждый раз будет выводиться предупреждение.

Итак, открываем файл, в цикле считываем его построчно посредством метода readline(). Как только будет прочитана пустая строка — выходим из цикла. Каждую строку разбиваем на составляющие, используя в качестве разделителя пробел (метод split). Если строка содержит на шестой позиции символы «"GET», а на девятой — «200» (нужный нам код ответа), то записываем в переменные седьмое поле, содержащее URL запрошенной страницы, и десятое, в котором указано количество переданных байт.

При обработке строки URL используется уже знакомый нам срез, чтобы отбросить все, что находится после символа «?», то есть получить просто имя страницы без параметров, которые передаются на сервер. Позицию знака «?» возвращает функция «index», а чтобы не возникало ошибки при отсутствии его в строке, предварительно проверяем его наличие функцией «find».

Немного подробнее остановимся на переменных stat и cnts. Это ассоциативные массивы, хранящие пары «ключ-значение». В роли ключа будет выступать имя страницы, а в роли значения — трафик и количество обращений.

Метод keys, примененный к ассоциативному массиву, возвращает список ключей; метод sort осуществляет сортировку списка.

Далее в цикле по всем элементам в statkeys результат выводится на экран и попутно вычисляются суммарные значения числа обращений и трафика.

Обратите внимание на цифры в конструкциях %s и %d — они указывают число символов, которое будет занимать переменная. По умолчанию выравнивание осуществляется по правому краю с помощью пробелов. Минус перед числом заставляет осуществлять выравнивание влево.

По результатам работы будет выведена таблица с указанием количества обращений к каждой странице и объема трафика.

Обработка лог-файла размером 17,5 Мбайт на Celeron 466 (128 Мбайт памяти) занимает около 10 секунд.

Сбор статистики работы канала Интернета

Задача

Каждые 15 минут требуется проверять работоспособность канала путем отсылки десяти пакетов размером 1400 байт. Статистику (количество принятых пакетов, среднее время переноса и среднюю девиацию stddev) будем записывать в лог-файле, а при отсутствии соединения — отправлять письмо администратору.

Решение

Цель данного примера — показать приемы взаимодействия Python с операционной системой. Для проверки работы канала будет использоваться системная утилита ping, отправка уведомления будет осуществляться с помощью команды mail, а логи будут записываться с помощью программы logger.

Для взаимодействия с программами операционной системы используются две функции модуля os — «popen» и «system». Первая из них открывает канал с указанной в качестве аргумента программой, что позволяет считать ее вывод с помощью метода readlines (или любого другого, который лучше подходит в той или иной ситуации). Функция «system» просто выполняет указанную команду и восстанавливает лишь код возврата.

Код сценария представлен ниже:

```
#!/usr/local/bin/python
import os
pingcmd = '/sbin/ping -q -c 10 -s 1400 '
defaulthost = 'ya.ru'
loggercmd = '/usr/bin/logger'
mailcmd = '/usr/bin/mail -s \'Ping notification\"
admin = 'admin@site.ru'
def tolog(msg, host):
    os.system(loggercmd + ' "ping ' + host + ': ' + msg + '"')
def tomail(msg):
    os.system('echo ' + msg + '|' + mailcmd + ' ' + admin)
def pingchannel(host=defaulthost):
    reply = os.popen(pingcmd + host).readlines()
    received = int(reply[3].split(' ')[3])
    if received == 0:
        tolog('%2d | %7s | %7s' % (received, '-', '-'), host)
        tomail('No ping to %s' % host)
    else:
        stat = reply[4].split(' ')[3].split('/')
        average = float(stat[1])
        stddev = float(stat[3])
        tolog('%2d | %7.3f | %7.3f' % (received, average, stddev),
host)
pingchannel()
pingchannel('samag.ru')
pingchannel('cam12.noping.ru')
```

Большинство используемых здесь команд применялись ранее. Поясним лишь конструкцию host=defaulthost в определении функции «pingchannel». Такая запись позволяет определить значение аргумента по умолчанию — если функция вызывается с аргументом (последние две строки), то применяется переданное значение. Если же аргумент при вызове функции опущен, то он принимает значение, указанное после знака равенства.

Подробности работы использованных в сценарии системных утилит (mail, logger, ping) смотрите на соответствующих страницах справочного руководства вашей операционной системы.

В результате работы рассмотренного сценария в системный журнал (согласно настройкам syslog) будут помещаться следующие строки:

```
Jun 17 16:07:06 host serg: ping ya.ru:          10 | 34.307 | 0.529
Jun 17 16:07:08 host serg: ping samag.ru:       9 | 11.772 | 0.058
Jun 17 16:07:21 host serg: ping cam12.noping.ru: 0 | — |
```

Если с какого-то из тестируемых узлов не будет получено ни одного ответа, администратору на электронный ящик будет отправлено сообщение об этом:

```
No ping to cam12.noping.ru
```

Данный сценарий можно периодически запускать по cron, и в итоге будет накапливаться статистика качества работы каналов. Лог-файл можно обрабатывать также сценарием на Python, основные принципы такой обработки были рассмотрены в примере 2.

Заключение

Приведенные выше примеры показывают, что Python может с успехом применяться для решения самого широкого круга задач администрирования. Он удобен для работы с файлами, хорошо обрабатывает текстовые документы (хотя в этом он несколько уступает мощи и гибкости Perl), предоставляет необходимые средства для взаимодействия с операционной системой.

Благодаря концепции повторного использования кода через некоторое время нарабатывается целая «библиотека» различных функций (например, функцию «pingchannel» из третьего примера можно будет в дальнейшем использовать и в других сценариях, нуждающихся в проверке работоспособности канала), и программировать становится все проще. А строгий синтаксис дает гарантию того, что через несколько лет вы сами (или пришедший вам на смену системный администратор) сможете без труда разобраться в вашем сценарии.