Программирование работы с SSH при помощи Paramiko (Перевод)

*OpenSSH - это вездесущий метод удалённого безопасного доступа к машине и передачи файлов. Многие - системные администраторы, инженеры автоматизации тестов, веб-разработчики и другие люди используют этот методы ежедневно. Написание скриптов для ssh на Python может быть тяжёлым занятием, но модуль Paramiko позволяет решить эту задачу проще.*

*Это репринт статьи, написанной для*[*Python Mag­a­zine*](http://www.pythonmagazine.com/)*в колонку Com­pletely Dif­fer­ent и опубликованной в октябрьском выпуске 2008 года. Тут она приводится в оригинальной форме, со всеми ошибками и т.д.*

SSH везде. OS X, Linux. Solaris и даже Windows предлагают OpenSSH серверы для удалённого доступа и передачи файлов. Он уже давно заменил такие методы удалённого доступа как tel­net и rlogin. И хотя эти системы ещё могут где-то применяться, их широкое использование ушло в прошлое вместе с быстрым принятием набора инструментов OpenSSH.

Сам OpenSSH - это набор инструментов, основанный на протоколе ssh2. Он содержит инструменты для удалённого безопасного входа (ssh), безопасной передачи файлов (scp и sftp) и инструмент управления ключами.

На большинстве ОС клиентские инструменты (ssh, scp, sftp) уже доступны для пользователя. Кроме того, пользователи могут легко установить и сконфигурировать серверные утилиты на той машине, к которой они хотят получить удалённый доступ.

Многие люди ежедневно пользуются OpenSSH и многие тратят уйму времени на попытки написать скрипты для автоматизации его использования. Большая часть этих скриптов оборачивает команды командной строки напрямую (ssh, scp и т.д.). Они используют инструменты типа Pexpect для получения паролей и пытаются работать напрямую с выводом исполняемых файлов.

Потратив на это кучу времени я пришёл, чтобы сказать Вам, что это приводит к ошибкам, сложностям в тестировании и всю эту машинерию очень тяжело поддерживать в рабочем состоянии.

Когда Вы пытаетесь работать с выводом утилит командной строки, следите за кодами завершения, жонглируете таймаутами - Вы на ложном пути. Вот для чего нужно Paramiko.

Я познакомился с Paramiko некоторое время назад. Он использует PyCrypto для предоставления Python'у интерфейса к протоколу ssh2. Этот модуль предоставляет всё, что Вы только можете желать, включая аутентификацию на основе ssh ключей, доступ к ssh-шелу и sftp.

После моего знакомства с Paramiko мой подход к использованию ssh изменился. Вместо изнурительных экспериметов работы с командной строкой, я получаю программный доступ к протоколу и ко всем его инструментам. При чём всё это в Python-стиле.

**О Paramiko**

Paramiko - это "чистый" модуль Python, который может быть легко установлен, как и любой другой модуль. Однако, PyCrypto написан большей частью на С, так что Вам может потребоваться компилятор, чтобы установить его на свою платформу.

Сам Paramiko имеет обширную документацию по API и активную рассылку. В качестве дополнительного бонуса, есть порт всего этого на Java (но давайте не будем об этом) если Вам понадобится сделать что-то на Java.

Paramiko так же предоставляет реализацию ssh и sftp для сервера. С полным набором функций. Я использую его в как в полноценных многопоточных приложениях, так и в ежедневных скриптах. Так же есть система установки и развёртывания под названием Fabric, тоже построенная на Paramiko, предоставляющая возможность развёртывания утилит через ssh.

**Начнём**

Главный класс API Paramiko - это "paramiko.SSHClient". Он предоставляет базовый интерфейс, который Вам поднадобится для установки соединения и передачи файлов. Вот простой пример:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4 | **import** paramiko  ssh = paramiko.SSHClient()  ssh.connect('127.0.0.1', username='jesse',  password='lol') |

Этот код создаёт новый объект SSHClient и затем вызывает метод connect() для подключения к локальному ssh серверу. Проще и быть не может!

**Host keys**

Один из сложнейших аспектов ssh-аутентификации - *host keys.*Когда Вы устанавливаете ssh соединения с удалённой машиной, ключ хоста автоматически сохраняется в файле в вашей домашней директории с названием .ssh/konwn\_hosts. Если Вы когда-нибудь подключались к новому хосту через ssh, видели сообщение вроде этого:

The authenticity of host 'localhost (::1)' can't be

established.

RSA key fingerprint is

22:fb:16:3c:24:7f:60:99:4f:f4:57:d6:d1:09:9e:28.

Are you sure you want to continue connecting

(yes/no)?

и печатали в ответ на вопрос “yes” — то Вы добавляли ключ в файл konwn\_hosts. Эти ключи важны потому как их принятие реализует уровень доверия между хостами. Если ключ был изменён и скомпроментирован каким-либо образом, то Ваш клиент отвергнет соединение даже не предупредив Вас об этом.

Paramiko использует то же правило. Вы должны принять и авторизовать использование и хранение этих ключей для каждого хоста. К счастью, вместо того, чтобы заниматься каждым ключом по отдельности, Вы можете настроить "волшебную" политику.

Поведение по умолчанию SSHClient'а - отвергать соединение с узлом ("paramiko.RejectPolicy"), чей ключ не сохранён в вашем файле known\_hosts. Это может быстро надоесть при работе в тестовом окружении, где машины то появляются, то уходят, и где Вам постоянно приходится переустанавливать системы.

Установка политики работы с ключами требует одного метода, который вызывается у объекта ssh клиента (set\_missing\_host\_key\_poliy()), который задаёт поведение, которое должны быть применено к несвязаным ключам. Если Вы, как и я, ленивы, то Вы будете использовать paramiko.AutoAddPolicy(), что приводит к автоматическому принятию неизвестных ключей.

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **import** paramiko  ssh = paramiko.SSHClient()  ssh.set\_missing\_host\_key\_policy(  paramiko.AutoAddPolicy())  ssh.connect('127.0.0.1', username='jesse',  password='lol') |

Конечно, не используйте эту политику если Вы работаете с машинами, которые Вам не известны или которым Вы не доверяете. Инструменты, построенные на Paramiko должны использовать эту чрезмерно либеральную политику лишь как опцию настройки.

**Запускаем простые команды**

После того, как мы подключились, мы можем запускать команды и получать их результат.

ssh использует тот же тип ввода / вывода / обработки ошибок, с которым Вы должны быть знакомы по другим UNIX-подобным приложениям. Ошибки посылаются на стандартный вывод ошибок, вывод - на стандартный вывод, а ввод берётся со стандартного ввода.

Итак, ответ от клиента возвращается в виде кортежа - (stdin, stdout, stderr) - где все три элемента - файлоподобные объекты, из которых Вы можете читать (или писать - в stdin). Например:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ...  >>> ssh.connect('127.0.0.1', username='jesse',  ... password='lol')  >>> stdin, stdout, stderr = \  ... ssh.exec\_command("uptime")  >>> type(stdin)  <class 'paramiko.ChannelFile'>  >>> stdout.readlines()  ['13:35 up 11 days, 3:13, 4 users, load averages: 0.14 0.18 0.16**\n**'] |

За кулисами Paramiko открывает новый объект ”paramiko.Channel”, который представляет безопасный тунель к удалённому хосту. Этот объект ведёт себя как обычный объект сокета в Python. Когда мы вызваем ”exec\_command()”, открывается канал к хосту, а обратно мы получаем ”paramiko.ChannelFile” файлоподобный объект, который представляет данные, посылаемые на и с удалённого хоста.

Одна из документированных "фитч" объекта ChannelFile состоит в том, что Вам всё время надо читать из stderr и stdout. Если удалённый хост пошлёт обратно достаточно информации, чтобы забить буфер, то хост подвиснет, ожидая, пока ваша программа считает посланные данные. Два способа справиться с этим - вызывать readlines(), как мы сделали выше, или read(). Если Вам нужен внутренний буфер - можно проводить итерацию при помощи readline().

Это была простая форма подключения и запуска команд для получения ответа. Для многих админских задач этого вполне достаточно, так как работая со строками в Python Вы можете достать всё, что Вам нужно. Но давайте теперь посмотрим на что-то с большим объёмом вывода, что ещё и требует пароль:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4 | ssh.connect('127.0.0.1', username='jesse',  password='lol')  stdin, stdout, stderr = ssh.exec\_command(  "sudo dmesg") |

Упс. Я только что запустил команду sudo. А ведь ей нужен пароль для удалённого хоста... Без проблем:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ssh.connect('127.0.0.1', username='jesse',  password='lol')  stdin, stdout, stderr = ssh.exec\_command(  "sudo dmesg")  stdin.write('lol**\n**')  stdin.flush()  data = stdout.read.splitlines()  **for** line **in** data:  **if** line.split(':')[0] == 'AirPort':  **print** line |

Вот так! Я удалённо залогинился и нашёл все сообщения от своей карты Air­port. Ключом к этому послужила запись пароля в "файл"  stdin, так что получив пароль, sudo меня распознал.

Если у Вас уже крутится на языке вопрос - да, таким образом Вы можете создать свою собственную интерактивную оболочку. Вы можете захотеть сделать что-то вроде этого для создания админской оболочки при помощи модуля cmd, чтобы управлять машинами в своей лаборатории.

Используя Paramiko сделать это становится просто. В листинге 1.1 показан простой способ достичь этого - мы оборачиваем манипуляции paramiko в методы RunCommand, позволяя пользователю добавлять столько хостов, сколько он захочет, соединяясь с ними и выполняя на них команды.

Листинг 1.1

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52 | *#!/usr/bin/python*    **import** paramiko  **import** cmd    **class** RunCommand(cmd.Cmd):  """ Simple shell to run a command on the host """    prompt = 'ssh > '    **def** \_\_init\_\_(self):  cmd.Cmd.\_\_init\_\_(self)  self.hosts = []  self.connections = []    **def** do\_add\_host(self, args):  """add\_host <host,user,password>  Add the host to the host list"""  **if** args:  self.hosts.append(args.split(','))  **else**:  **print** "usage: host <hostip,user,password>"    **def** do\_connect(self, args):  """Connect to all hosts in the hosts list"""  **for** host **in** self.hosts:  client = paramiko.SSHClient()  client.set\_missing\_host\_key\_policy(  paramiko.AutoAddPolicy())  client.connect(host[0],  username=host[1],  password=host[2])  self.connections.append(client)    **def** do\_run(self, command):  """run <command>  Execute this command on all hosts in the list"""  **if** command:  **for** host, conn **in** zip(self.hosts, self.connections):  stdin, stdout, stderr = conn.exec\_command(command)  stdin.close()  **for** line **in** stdout.read().splitlines():  **print** 'host: %s: %s' % (host[0], line)  **else**:  **print** "usage: run <command>"    **def** do\_close(self, args):  **for** conn **in** self.connections:  conn.close()    **if** \_\_name\_\_ == '\_\_main\_\_':  RunCommand().cmdloop() |

Вот пример вывода:

ssh > add\_host 127.0.0.1,jesse,lol

ssh > connect

ssh > run uptime

host: 127.0.0.1: 14:49 up 11 days, 4:27, 8 users,

load averages: 0.36 0.25 0.19

ssh > close

Это всего лишь proof-of con­cept псевдо-интерактивной оболочки. Вот что Вы ещё можете к ней добавить:

* улучшенное отображение многострочного вывода
* обработчик стандартных ошибок
* что-нибудь в метод выхода
* поток для выполнения команды / возврата результата

Как и когда дело касается шелла границей тут будет только ваша фантазия. Такие инструменты, как pssh, osh, Fabric и т.д. все отображают данные по разному и каждый из них по своему группирует вывод от разных хостов.

**Отправка и получение файлов**

Работа с файлами в Paramiko управляется реализацией sftp, и, как и работа с клиентом ssh, она проста как два пальца.

Мы начинаем с создания нового paramiko.SSHClient, как и раньше:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | **import** paramiko  ssh = paramiko.SSHClient()  ssh.set\_missing\_host\_key\_policy(  paramiko.AutoAddPolicy())  ssh.connect('127.0.0.1', username='jesse',  password='lol') |

Но на этот раз мы вызовем метод ”open\_sftp()” после установки соединения. ”open\_sftp()” возвращает объект клиента ”paramiko.SFTPClient”, который поддерживает все стандартный операции sftp (stat, put, get и т.д.). В нашем примере мы воспользуемся операцией “get”, чтобы скачать файл ”remotefile.py” с удалённой машины и сохранить его на локальной машине под именем ”localfile.py”.

ftp = ssh.open\_sftp()  
ftp.get('remotefile.py', 'localfile.py')  
ftp.close()

Отправка файла на удалённую машину (операция “put”) работает точно так же. Мы лишь меняем порядок аргументов:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3 | ftp = ssh.open\_sftp()  ftp.get('localfile.py', 'remotefile.py')  ftp.close() |

То, что мне нравится в sftp клиенте, реализованном в Paramiko, так это то, что он поддерживает такие операции как stat, chmod, chown и т.д. Очевидно, что они могут работать по разному на разных удалённых машинах, так как не все сервера реализуют протокол целиком, но всё равно их наличие - большой плюс.

Вы можете запросто написать функцию типа glob.glob() для просмотра файлового дерева удалённой системы в поисках файлов, отвечающих заданному шаблону. Можно проводить поиск на основе разрешений, размера файлов и т.д.

Однако, стоит заметить (и я часто с этим сталкивался): sftp, как протокол, гораздо более ограничен, чем scp. scp позволяет использовать шаблоны UNIX в имени файла при получении его с удалённой машины. А вот sftp ожидает от Вас полного пути к файлу, который Вы хотите скачать. Вот пример:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1 | ftp.get('\*.py', '.') |

В большинстве случаев это значит "скачать все файлы с расширением .py в локальную директорию на моей машине". Но, к сожалению, sftp Вас не поймёт (см листинг 2). Мне это знание досталось тяжёлым путём, после того, как я потратил несколько часов на разбор реализации клиента sftp.

Листинг 2:

[**?**](http://www.ericbess.com/ericblog/2008/03/03/wp-codebox/#examples)[**View Code**](http://www.blogger.com/blogger.g?blogID=2142461056765545912)**PYTHON**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | >>> ftp.get("./\*.py", '.')  Traceback (most recent call last):  File "<stdin>", line 1, **in** <module>  File "/Library/Python/2.5/site-packages/paramiko/sftp\_client.py",  line 567, **in** get  fr = self.file(remotepath, 'rb')  File "/Library/Python/2.5/site-packages/paramiko/sftp\_client.py",  line 238, **in** open  t, msg = self.\_request(CMD\_OPEN, filename, imode, attrblock)  File "/Library/Python/2.5/site-packages/paramiko/sftp\_client.py",  line 589, **in** \_request  **return** self.\_read\_response(num)  File "/Library/Python/2.5/site-packages/paramiko/sftp\_client.py",  line 636, **in** \_read\_response  self.\_convert\_status(msg)  File "/Library/Python/2.5/site-packages/paramiko/sftp\_client.py",  line 662, **in** \_convert\_status  **raise** IOError(errno.ENOENT, text)  IOError: [Errno 2] No such file |

**В заключение**

Я надеюсь, что Вы увидели достаточно, чтобы оценить Paramiko. Это одна из тех жемчужин в сообществе Python, которая помогает мне в ежедневной работе. Я могу программировать удалённое администрирование, писать тестовые плагины, которые легко исполняют удалённые операции, и всё это без необходимости устанавливать дополнительных демонов на удалённой машине.

SSH везде, и раньше или позже Вы тоже столкнётесь  необходимостью писать под него программы. Так почему бы сразу не сэкономить себе время и не воспользоваться Paramiko?