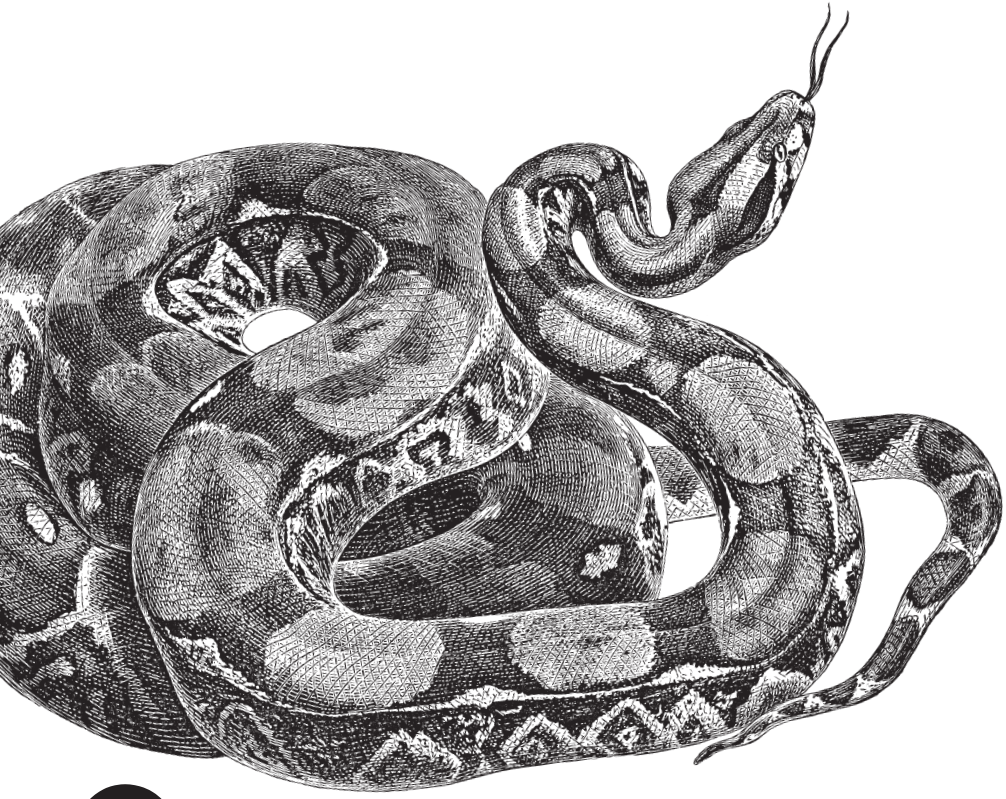
Эффективное решение проблем с помощью языка Python

Python

в системном администрировании UNIX и Linux



СИМВОЛ®

O'REILLY®

Ноа Гифт, Джереми М. Джонс

Python

for Unix and Linux System Administration

Noah Gift, Jeremy M. Jones

O'REILLY®

Python

в системном администрировании UNIX и Linux

Ноа Гифт и Джереми М. Джонс

Санкт-Петербург — Москва
2009

Ноа Гифт и Джереми М. Джонс

Python в системном администрировании UNIX и Linux

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Редактор	*Ю. Бочина*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Гифт Н., Джонс Д.
Python в системном администрировании UNIX и Linux – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 512 с., ил.
ISBN 978-5-93286-149-3

Книга «Python в системном администрировании UNIX и Linux» демонстрирует, как эффективно решать разнообразные задачи управления серверами UNIX и Linux с помощью языка программирования Python. Каждая глава посвящена определенной задаче, например многозадачности, резервному копированию данных или созданию собственных инструментов командной строки, и предлагает практические методы ее решения на языке Python. Среди рассматриваемых тем: организация ветвления процессов и передача информации между ними с использованием сетевых механизмов, создание интерактивных утилит с графическим интерфейсом, организация взаимодействия с базами данных и создание приложений для Google App Engine. Кроме того, авторы книги создали доступную для загрузки и свободно распространяемую виртуальную машину на базе Ubuntu, включающую исходные тексты примеров из книги и способную выполнять примеры, использующие SNMP, IPython, SQLAlchemy и многие другие утилиты.

Издание рассчитано на широкий круг специалистов – всех, кто только начинает осваивать язык Python, будь то опытные разработчики сценариев на языках командной оболочки или относительно мало знакомые с программированием вообще.

ISBN 978-5-93286-149-3
ISBN 978-0-596-51582-9 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 12.01.2009. Формат 70×100¹/₁₆. Печать офсетная.
Объем 32 печ. л. Тираж 1000 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Я посвящаю эту книгу доктору Джозефу Е. Богену (Joseph E. Bogen),
моей матушке и моей супруге Леа – трем людям,
которые любили меня и верили в меня больше всех.

Ноа

Я посвящаю эту книгу моей жене Дебре и моим детям,
Зейну и Юстусу. Вы вдохновляли меня, дарили мне
свои улыбки и проявляли величайшее терпение,
пока я работал над этой книгой. Она по праву может
считаться настолько же вашей, насколько и моей.
Джереми

Оглавление

Предисловие 11

Введение 12

1. Введение 21

Почему Python? 21
Мотивация 28
Основы 29
Выполнение инструкций в языке Python 30
Использование функций в языке Python 35
Повторное использование программного кода с помощью инструкции import 39

2. IPython 45

Установка IPython 46
Базовые понятия 48
Справка по специальным функциям 56
Командная оболочка UNIX 61
Сбор информации 81
Автоматизация и сокращения 95
В заключение 101

3. Текст 102

Встроенные компоненты Python и модули 103
Анализ журналов 146
ElementTree 153
В заключение 158

4. Создание документации и отчетов 159

Автоматизированный сбор информации 160
Сбор информации вручную 163
Форматирование информации 174
Распространение информации 180
В заключение 185

5. Сети 186

Сетевые клиенты 186
Средства вызова удаленных процедур 199
SSH 206
Twisted 209
Scapy 216
Создание сценариев с использованием Scapy 219

6. Данные 221

Введение 221
Использование модуля OS для взаимодействия с данными 222
Копирование, перемещение, переименование и удаление данных 224
Работа с путями, каталогами и файлами 226
Сравнение данных 230
Объединение данных 233
Поиск файлов и каталогов по шаблону 239
Обертка для rsync 241
Метаданные: данные о данных 244
Архивирование, сжатие, отображение и восстановление 246
Использование модуля tarfile для создания архивов TAR 246
Использование модуля tarfile для проверки содержимого файлов TAR 249

7. SNMP 252

Введение 252
Краткое введение в SNMP 252
IPython и Net-SNMP 256
Исследование центра обработки данных 260
Получение множества значений с помощью SNMP 263
Создание гибридных инструментов SNMP 270
Расширение возможностей Net-SNMP 271
Управление устройствами через SNMP 275
Интеграция SNMP в сеть предприятия с помощью Zenoss 276

8. Окрошка из операционных систем 278

Введение 278
Кросс-платформенное программирование на языке Python в UNIX 279
PyInotify 291
OS X 293
Администрирование систем Red Hat Linux 298
Администрирование Ubuntu 299
Администрирование систем Solaris 299

Виртуализация 300
Облачная обработка данных 301
Использование Zenoss для управления серверами Windows из Linux 309

9. Управление пакетами 313

Введение 313
Setuptools и пакеты Python Eggs 314
Использование easy_install 315
Дополнительные особенности easy_install 318
Создание пакетов 324
Точки входа и сценарии консоли 329
Регистрация пакета в Python Package Index 330
Distutils 332
Buildout 335
Использование Buildout 335
Разработка с использованием Buildout 339
virtualenv 339
Менеджер пакетов EPM 344

10. Процессы и многозадачность 350

Введение 350
Модуль subprocess 350
Использование программы Supervisor для управления процессами 361
Использование программы screen для управления процессами 364
Потоки выполнения в Python 365
Процессы 378
Модуль processing 379
Планирование запуска процессов Python 382
Запуск демона 384
В заключение 388

11. Создание графического интерфейса 390

Теория создания графического интерфейса 390
Создание простого приложения PyGTK 392
Создание приложения PyGTK для просмотра файла журнала веб-сервера Apache 394
Создание приложения для просмотра файла журнала веб-сервера Apache с использованием curses 398
Веб-приложения 403
Django 404
В заключение 426

12. Сохранность данных 427

Простая сериализация 428
Реляционная сериализация 448
В заключение 458

13. Командная строка 459

Введение 459
Основы использования потока стандартного ввода 460
Введение в optparse 462
Простые шаблоны использования optparse 462
Внедрение команд оболочки в инструменты командной строки на языке Python 470
Интеграция конфигурационных файлов 477
В заключение 479

14. Практические примеры 480

Управление DNS с помощью сценариев на языке Python 480
Использование протокола LDAP для работы с OpenLDAP, Active Directory и другими продуктами из сценариев на языке Python 482
Составление отчета на основе файлов журналов Apache 484
Зеркало FTP 492

Приложение. Функции обратного вызова 496

Алфавитный указатель 499

Вступительное слово

Я была приятно взволнована предложением выполнить предварительный обзор книги, посвященной использованию языка Python для нужд системного администрирования. Я вспомнила свои ощущения, когда впервые познакомилась с языком Python после многих лет программирования на других языках; это было похоже на свежесть весеннего ветра и согревающее тепло солнца после долгой зимы. Программирование на этом языке оказалось настолько необычайно простым и увлекательным делом, что мне удавалось заканчивать программы намного раньше, чем прежде.

Будучи системным администратором, я использовала язык Python в основном для решения задач системного и сетевого администрирования. Я заранее знала, насколько востребованной будет хорошая книга, посвященная применению языка Python в системном администрировании, и рада сказать, что это в полной мере относится к данной книге. Авторам, Ноа (Noah) и Джереми (Jeremy), удалось написать интересную и умную книгу о языке Python, который прочно обосновался в сфере системного администрирования. Я нахожу эту книгу полезной и увлекательной.

Две первые главы представляют собой введение в язык программирования Python для системных администраторов (и других), которые еще не знакомы с ним. Я отношу себя к программистам на языке Python среднего уровня, поэтому немало нового узнала из этой книги. Я полагаю, что даже искушенные программисты найдут здесь несколько новых приемов. Особенно я рекомендую прочитать главы, посвященные сетевому администрированию и управлению сетевыми службами, SNMP и управлению гетерогенными сетями, потому что в центре их внимания находятся нетривиальные и реальные задачи, с которыми системные администраторы сталкиваются ежедневно.

Элин Фриш (Æleen Frisch), июль 2008

Предисловие

Типографские соглашения

В этой книге используются следующие типографские соглашения:

Курсив

Курсивом выделяются новые термины, адреса URL, адреса электронной почты, имена файлов и их расширения.

`Моноширинный шрифт`

Используется для оформления листингов программ, для обозначения в тексте таких программных элементов, как имена переменных или функций, баз данных, типов данных, переменных окружения, инструкций, ключевых слов, утилит и модулей.

`Моноширинный жирный шрифт`

Используется для выделения команд и другого текста, который должен вводиться пользователем.

`Моноширинный курсив`

Используется для выделения текста, который пользователь должен заменить своими значениями или значениями, определяемыми контекстом.

Таким способом выделяются советы, предложения и примечания общего характера.

Таким способом выделяются предупреждения и предостережения.

Использование программного кода примеров

Данная книга призвана оказать вам помощь в решении ваших задач. Вообще вы можете свободно использовать примеры программного кода из этой книги в своих программах и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за раз-

решением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Для цитирования данной книги или примеров из нее при разъяснении каких-либо вопросов получение разрешения не требуется. При включении существенных объемов программного кода примеров из этой книги в документацию на вашу продукцию вам *необходимо* получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: «Python for Unix and Linux System Administration by Noah Gift and Jeremy M. Jones. Copyright 2008 Noah Gift and Jeremy M. Jones, 978-0-596-51582-9».

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу: *permissions@oreilly.com*.

Safari® Books Online

Если на обложке технической книги есть пиктограмма «Safari® Books Online», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Отзывы и предложения

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в Соединенных Штатах Америки или в Канаде)
707-829-0515 (международный)
707-829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на сайте книги:

http://www.oreilly.com/9780596515829

Свои пожелания и вопросы технического характера отправляйте по адресу:

bookquestions@oreilly.com

Дополнительную информацию о книгах, обсуждения, Центр ресурсов издательства O'Reilly вы найдете на сайте:

http://www.oreilly.com

Благодарности

От Ноа

Лист благодарностей этой книги я хочу начать с доктора Джозефа Е. Богена (Joseph E. Bogen), потому что он – человек, который оказал наибольшее влияние на меня в тот момент, когда это было больше всего необходимо. Я встретил доктора Богена, когда работал в фирме Caltech, и он открыл мне глаза на другой мир, давая советы по жизненным ситуациям, психологии, неврологии, математике, по исследованиям в области сознания и во многих других областях. Это был умнейший человек, которого я когда-либо встречал, и я искренне любил его. Когда-нибудь я напишу книгу об этом опыте. Я опечален тем, что он не сможет прочитать ее; его смерть стала для меня большой утратой.

Я хочу сказать спасибо моей жене Леа (Leah), самой лучшей из всех женщин, встречавшихся мне. Без твоей любви и поддержки мне не удалось бы написать эту книгу. Ты терпелива, как ангел. Я надеюсь и дальше идти с тобой по жизни, я люблю тебя. Я также хочу поблагодарить моего сына Лиама (Liam), которому полтора года, за то, что терпел, пока я работал над этой книгой. Мне пришлось сильно урезать наши занятия музыкой и спортом, поэтому я должен вернуть тебе в два раза больше, мой маленький козленок.

Моей матушке: я люблю тебя и хочу сказать спасибо, что подбадривала меня все время.

Конечно же, я хочу сказать спасибо Джереми М. Джонсу (Jeremy M. Jones), моему соавтору – за то, что согласился написать эту книгу вместе со мной. Я думаю, из нас получилась отличная команда. У нас разные стили, но они прекрасно дополняют друг друга, и мы написали хорошую книгу. Вы дали мне много новых знаний о языке Python и были для меня хорошим партнером и другом. Спасибо!

Титус Браун (Titus Brown), которого теперь я должен бы называть доктором Брауном, был тем, кто разжег во мне первый интерес к языку Python, когда я встретил его в фирме Caltech. Это еще один пример того, какое важное значение может иметь один человек, и я рад считать его своим «старым» другом, которого не купишь ни за какие деньги. Он не уставал спрашивать меня: «Почему ты не используешь Python?». И однажды я решил попробовать. Если бы не Титус, я безусловно вернулся бы обратно к языкам Java и Perl. Вы можете почитать его блог по адресу: *http://ivory.idyll.org/blog*.

У Шеннона Беренса (Shannon Behrens) золотая голова, острый, как бритва, ум и потрясающее знание языка Python. Я познакомился

с Шенноном благодаря Титусу, и мы быстро подружились с ним. Шеннон – настоящий человек дела, во всех смыслах этого слова, и он дал мне огромный объем знаний о языке Python, можно даже сказать, гигантский. Его помощь во всем, что касалось языка Python, и в редактировании этой книги была просто неоценима, и я чрезвычайно обязан ему за это. Иногда я с ужасом думаю, какой могла бы быть эта книга без его помощи. Я не могу себе представить компанию, которая может упустить его, и я надеюсь помочь ему с его первой книгой. Наконец, он просто удивительный технический рецензент. Вы можете почитать его блог по адресу: *http://jjinux.blogspot.com/*.

Еще одним звездным техническим рецензентом был Дуг Хеллманн (Doug Hellmann). Сотрудничество с ним было исключительно плодотворным и полезным. Джереми и мне необычайно повезло в том, что нам удалось заполучить специалиста такого масштаба в качестве рецензента. Он не ограничился своим служебным долгом и стал настоящей движущей силой. Он был для нас неиссякаемым источником вдохновения, пока мы вместе с ним работали в компании Racemi. Вы можете почитать его блог по адресу: *http://blog.doughellmann.com/*.

Кому еще я хотел бы выразить свою признательность?

Скотту Лирсину (Scott Leerseen) – за обзор книги и за полезные советы в процессе работы над ней. Я получал огромное удовольствие от жарких споров, разгоравшихся вокруг фрагментов программного кода. Но помните – я всегда прав.

Альфредо Деза (Alfredo Deza) – за работу над настройкой виртуальной машины с Ubuntu, которая была необходима для работы над книгой. Твой опыт был для нас очень ценным.

Лайзе Дейли (Liza Daly) – за отзывы к первым черновым наброскам некоторых частей этой книги. Они были чрезвычайно полезными.

Джеффу Рашу (Jeff Rush) – за помощь и советы в работе с Buildout, Eggs и Virualenv.

Аарону Хиллегассу (Aaron Hillegass), владельцу замечательной обучающей компании Big Nerd Ranch, – за ценные советы и помощь во время работы. Мне крупно повезло, что посчастливилось встретиться с ним.

Марку Лутцу (Mark Lutz), под руководством которого я прошел курс обучения языку Python и который написал несколько замечательных книг по языку Python.

Членам сообщества Python в Атланте и участникам проекта PyAtl: *http://pyatl.org* – вы многому научили меня. Рик Коупленд (Rick Copeland), Рик Томас (Rick Thomas), Брендон Родс (Brandon Rhodes), Дерек Ричардсон (Derek Richardson), Джонатан Ла Кур (Jonathan La Cour), известный также под псевдонимом Mr. Metaclass, Дрю Смазерс (Drew Smathers), Кари Халл (Cary Hull), Бернард Меттьюс (Bernard Mat-

thews), Майкл Лангфорд (Michael Langford) и многие другие, кого я забыл упомянуть. Брендон и Рик Коупленд (Brandon and Rick Copeland) были в особенности полезны; они являются высококлассными программистами на языке Python. Вы можете почитать блог Брендона по адресу: *http://rhodesmill.org/brandon/*.

Григу Георгиу (Grig Gheorghiu) – за то, что делился с нами опытом системного администратора, за проверку советов и за то, что поддавал нам пинка, когда это было необходимо.

Моему работодателю, главному техническому директору и основателю компании Racemi, Чарльзу Уатту (Charles Watt). Я многому научился у вас и был рад, что вы знаете, когда какие кнопки нажимать. Помните, что я всегда готов написать для вас программу, пробежать 26-мильную дистанцию или проехать 200 миль на велосипеде – только сообщите мне, где и когда.

Доктору Нанде Ганесан (Nanda Ganesan), моему наставнику в аспирантуре Калифорнийского государственного университета в городе Лос-Анджелес (CSULA). Вы многому научили меня в области информационных технологий и в жизни и, кроме того, побуждали меня мыслить самостоятельно.

Доктору Синди Хейсс (Cindy Heiss) – моему профессору в мою бытность студентом. Вы приобщили меня к веб-разработке, научили верить в свои силы и, в конечном счете, оказали влияние на мою жизнь, спасибо!

Шелдону Блокбургеру (Sheldon Blockburger), позволившему мне попробовать свои силы в десятиборье в Калифорнийском государственном политехническом университете в городе Сан Луис Обиспо (Cal Poly SLO). Даже при том, что я не стал членом команды, вы развили во мне живой дух соперничества, качества борца и научили самодисциплине, предоставив мне самому отрабатывать забеги на короткие дистанции. И поныне еженедельные тренировки позволяют мне не потерять форму, в том числе и как программисту.

Брюсу Дж. Беллу (Bruce J. Bell), с которым я работал в Caltech. В течение нескольких лет совместной работы он учил меня программированию, делился своими знаниями операционной системы UNIX, и я очень признателен ему за это. С его статьями вы можете познакомиться по адресу: *http://www.ugcs.caltech.edu/~bruce/*.

Альберто Валезу (Alberto Valez), моему боссу в Sony Imageworks, – за то, что он был, пожалуй, лучшим боссом из всех, кто у меня когда-либо был, и за то, что предоставил мне возможность полностью автоматизировать мою работу.

Монтажеру фильмов Эду Фуллеру (Ed Fuller), который помогал мне советами и оставался отличным другом все это время.

Было много и других людей, оказывавших мне неоценимую помощь в работе, включая Дженнифер Девис (Jennifer Davis), еще одного друга по Caltech, которая предоставила несколько ценных отзывов; нескольких друзей и коллег по работе в компании Turner – Дуга Уэйка (Doug Wake), Уэйна Бланкарда (Wayne Blanchard), Сэма Олгуда (Sam Allgood), Дона Воравонга (Don Voravong); моих друзей и коллег по работе в Disney Feature animation, включая Шина Сомероффа (Sean Someroff), Грега Нигла (Greg Neagle) и Бобби Ли (Bobby Lea). Грег Нигл (Greg Neagle), в частности, очень многому меня научил в Mac OS X. Спасибо также Дж. Ф. Паниссету (J. F. Panisset), с которым я встретился в Sony Imageworks, учившему меня общим принципам разработки. И хотя теперь он главный технический директор, он мог бы считаться ценным кадром в любой компании.

Я хотел бы поблагодарить еще несколько человек, оказавших существенное содействие: Майка Вагнера (Mike Wagner), Криса МакДауэлла (Chris McDowell) и Шона Смута (Shaun Smoot).

Спасибо членам сообщества Python. В первую очередь спасибо Гвидо Ван Россуму (Guido van Rossum) за создание такого замечательного языка, за его качества настоящего лидера и за то, что был терпелив со мной, когда я обращался за советом по поводу этой книги. В сообществе Python есть большое число других знаменитостей, разрабатывающих инструменты, которыми я пользуюсь каждый день. Это Ян Бикинг (Ian Bicking), Фернандо Перез (Fernando Perez) и Вилле Вайнио (Ville Vainio), Майк Байер (Mike Bayer), Густаво Немейер (Gustavo Niemeyer) и другие. Спасибо Дэвиду Бизели (David Beazely) за его замечательную книгу и его фантастическое руководство «PyCon 2008 on Generators». Спасибо всем, кто пишет о языке Python и о системном администрировании. Ссылки на их работы вы сможете отыскать на странице *http://wiki.python.org/moin/systems_administration*. Спасибо также команде проекта Repoze: Тресу Сиверу (Tres Seaver) и Крису МакДонаху (Chris McDonough) (*http://repoze.org/index.html*).

Отдельная благодарность Филиппу Дж. Эби (Phillip J. Eby) за замечательный набор инструментальных средств, за проявленное терпение и советы по разделу, посвященному использованию библиотеки setuptools. Спасибо также Джиму Фултону (Jim Fulton) за то, что терпеливо отвечал на шквал моих вопросов по использованию ZODB и buildout. Особое спасибо Мартьяну Фассену (Martijn Fassen), который учил меня пользоваться такими продуктами, как ZODB и Grok. Если вам интересно заглянуть в будущее разработки веб-приложений на языке Python, обратите внимание на проект Grok: *http://grok.zope.org/*.

Спасибо сотрудникам журнала «Red Hat Magazine» – Джулии Брис (Julie Bryce), Джессике Гербер (Jessica Gerber), Баша Харрису (Bascha Harris) и Рут Сьюл (Ruth Suehle) за то, что позволили мне опробовать идеи, излагаемые в книге, в форме статей. Спасибо также Майку Мак-

Крери (Mike McCrery) из IBM Developerworks за то, что предоставил мне возможность опубликовать в форме статей некоторые идеи из книги.

Я хочу поблагодарить множество людей, которые в разные моменты моей жизни говорили, что мне что-то не по силам. Почти на каждом жизненном этапе я встречал людей, которые пытались отговорить меня: начиная с того, что я не смогу поступить в колледж, в который хотел бы поступить, и заканчивая тем, что я никогда не смогу изучать программирование. Спасибо вам за то, что давали мне дополнительный толчок к воплощению моих мечтаний. Люди способны выстроить свою жизнь, если они по-настоящему верят в себя; я мог бы посоветовать каждому пытаться сделать то, что он действительно хочет сделать.

Наконец, спасибо издательству O'Reilly и Татьяне Апанди (Tatiana Apandi) за то, что верили в мою способность написать книгу о применении языка Python в системном администрировании. Вы рискнули, поверили в меня и в Джереми, и я благодарю вас за это. Пусть ближе к концу книги Татьяна оставила издательство, чтобы воплотить свои мечты, тем не менее, мы продолжали чувствовать ее присутствие. Я также хотел бы отметить нового редактора Джулию Стил (Jilie Steele), которая была благожелательна и отзывчива. Вы привнесли целое море спокойствия, что лично я ценил очень высоко. В будущем я надеюсь еще услышать приятные новости от Джулии и буду счастлив снова работать с ней.

От Джереми

Длинный список благодарностей от Ноа заставил меня почувствовать себя неблагодарным человеком, потому что мой список не получится таким длинным, и обескураженным, так как он поблагодарил почти всех, кому мне тоже хотелось бы сказать спасибо.

В первую очередь я хотел бы вознести слова благодарности Господу Богу, с помощью которого я могу творить и без которого я ничего не смог бы сделать.

А в земном смысле в первую очередь я хотел бы поблагодарить мою супругу Дебру (Debra). Ты занимала детей другими делами, пока я работал над книгой. Ты сделала законом фразу: «Не беспокойте папу, он работает над своей книгой». Ты подбадривала меня, когда мне это было необходимо, и ты выделила мне пространство, которое мне так требовалось. Спасибо тебе. Я люблю тебя. Без тебя я не смог бы написать эту книгу.

Я также хотел поблагодарить моих славных детей, Зейна (Zane) и Юстуса (Justus) за их терпение по отношению к моей работе над книгой. Я пропустил большое число поездок с вами в парк Каменная Гора. Я по-прежнему укладывал вас спать, но я не оставался и не засыпал вместе с вами, как обычно. Последние несколько недель я пропускал шоу «Kid's Rock», которое выходит вечером по средам. Я пропустил так много, но вы терпеливо выдержали все это. Спасибо вам за ваше

терпение. И спасибо вам за то, что радовались, когда услышали, что я почти закончил книгу. Я люблю вас обоих.

Я хочу поблагодарить своих родителей, Чарльза и Линду Джонс (Charles and Linda Jones), за их поддержку моей работы над этой книгой. Но больше всего я хочу сказать им спасибо за то, что были для меня примером этики, за то, что научили меня работать над собой и с умом тратить деньги. Надеюсь, что все это я смогу передать своим детям, Зейну и Юстусу.

Спасибо Ноа Гифту (Noah Gift), моему соавтору, за то, что втянул меня в это дело. Оно оказалось тяжелым, тяжелее, чем я думал, и определенно одно из самых тяжелых, которое мне когда-либо приходилось делать. Когда вы работаете с человеком над чем-то, подобным книге, и под конец по-прежнему считаете его своим другом, я думаю, это достаточно характеризует его. Спасибо, Ноа. Эта книга не состоялась бы без тебя.

Я хочу поблагодарить нашу команду рецензентов. Ноа уже поблагодарил всех вас, но я хочу еще раз поблагодарить Дуга Хеллмана (Doug Hellman), Дженнифер Девис (Jennifer Davis), Шеннона Дж. Беренса (Shannon J. Behrens), Криса МакДауэлла (Chris McDowell), Титуса Брауна (Titus Brown) и Скотта Лирсина (Scott Leerseen). Вы удивительные люди. Бывали моменты, когда я заходил в тупик, и вы направляли мои мысли в нужное русло. Вы привнесли свое видение и помогли мне увидеть книгу с разных точек зрения. (В основном это относится к вам, Дженнифер. Если глава, посвященная обработке текста, принесет пользу системным администраторам, то только благодаря вам.) Спасибо вам всем.

Я хотел бы сказать спасибо нашим редакторам, Татьяне Апанди (Tatiana Apandi) и Джулии Стил (Julie Steele). Вы взяли на себя рутинный труд, освободив нас для работы над книгой. Вы обе облегчили нашу ношу.

Я также хочу выразить свою признательность Фернандо Перезу (Fernando Perez) и Вилле Вайнио (Ville Vainio) за потрясающие отзывы. Надеюсь, что мне удалось воздать должное IPython. И спасибо вам за IPython. Без него моя жизнь оказалась бы труднее.

Спасибо вам, Дункан МакГреггор (Duncan McGreggor), за помощь с примерами использования платформы Twisted. Ваши комментарии были чрезвычайно полезны. И спасибо, что вы продолжаете работать над этой замечательной платформой. Я надеюсь, что теперь буду использовать ее более широко.

Я благодарю Брема Мулинаара (Bram Moolenaar) и всех тех, кто когда-либо работал над редактором Vim. Почти все слова и теги XML, которые мне пришлось написать, были написаны с помощью Vim. В процессе работы над книгой я узнал несколько новых приемов и ввел их

в свой повседневный обиход. Редактор Vim позволил мне поднять мою производительность. Спасибо вам.

Я также хочу сказать спасибо Линусу Торвальдсу (Linus Torvalds), разработчикам Debian, разработчикам Ubuntu и всем тем, кто когда-либо работал над операционной системой Linux. Почти каждое слово, которое я напечатал, было напечатано в Linux. Вы обеспечили невероятную простоту настройки новых окружений и проверку различных идей. Спасибо вам.

Наконец, но ни в коем случае не меньше других, я хочу поблагодарить Гвидо ван Россума (Guido van Rossum) и всех тех, кто когда-либо работал над языком программирования Python. Я извлекал выгоду из вашего труда на протяжении нескольких последних лет. Два своих последних места работы я получил благодаря знанию языка Python. Язык Python и сообщество его поклонников не раз радовали меня с тех пор, как я начал использовать этот язык где-то в 2001–2002 годах. Спасибо вам. Python пришелся мне по душе.

1

Введение

Почему Python?

Если вы системный администратор, вам наверняка пришлось сталкиваться с Perl, Bash, ksh и некоторыми другими языками сценариев. Вы могли даже использовать один или несколько языков в своей работе. Языки сценариев часто позволяют выполнять рутинную, утомительную работу со скоростью и надежностью, недостижимой без них. Любой язык – это всего лишь инструмент, позволяющий выполнить работу. Ценность языка определяется лишь тем, насколько точно и быстро с его помощью можно выполнить свою работу. Мы считаем, что Python представляет собой ценный инструмент именно потому, что он дает возможность эффективно выполнять нашу работу.

Можно ли сказать, что Python лучше, чем Perl, Bash, Ruby или любой другой язык? На самом деле очень сложно дать такую качественную оценку, потому что всякий инструмент очень тесно связан с образом мышления программиста, использующего его. Программирование – это субъективный и очень личностный вид деятельности. Язык становится превосходным, только если он полностью соответствует потребностям программиста. Поэтому мы не будем доказывать, что язык Python лучше, но мы объясним причины, по которым мы считаем Python лучшим выбором. Мы также объясним, почему он так хорошо подходит для решения задач системного администрирования.

Первая причина, по которой мы считаем Python превосходным языком, состоит в том, что он очень прост в изучении. Если язык не способен быстро превратиться для вас в эффективный инструмент, его привлекательность резко падает. Неужели вы хотели бы потратить недели или месяцы на изучение языка, прежде чем вы окажетесь в состоянии написать на нем что-либо стоящее? Это особенно верно для системных администраторов. Если вы – системный администратор, проблемы могут

накапливаться быстрее, чем вы можете разрешать их. С помощью языка Python вы сумеете начать писать полезные сценарии буквально спустя несколько часов, а не дней или недель. Если язык не позволяет достаточно быстро приступить к написанию сценариев, это повод задуматься в целесообразности его изучения.

Однако язык, пусть и простой в изучении, но не позволяющий решать сложные задачи, также не стоит потраченных на него усилий. Поэтому вторая причина, по которой мы считаем Python превосходным языком программирования, заключается в том, что он позволяет решать такие сложные задачи, какие только можно вообразить. Вам требуется строку за строкой просматривать файлы журналов, чтобы выудить из них какую-то важную информацию? Язык Python в состоянии помочь решить эту задачу. Или вам требуется просмотреть файл журнала, извлечь из него определенные записи и сравнить обращения с каждого IP-адреса в этом файле с обращениями в каждом из файлов журналов (которые хранятся в реляционной базе данных) за последние три месяца, а затем сохранить результаты в реляционной базе данных? Вне всяких сомнений это можно реализовать на языке Python. Язык Python используется для решения весьма сложных задач, таких как анализ генных последовательностей, для обеспечения работоспособности многопоточных веб-серверов и сложнейших статистических вычислений. Возможно, вам никогда не придется решать подобные задачи, но будет совсем нелишним знать, что в случае необходимости язык поможет вам решать их.

Кроме того, если вы в состоянии выполнять сложнейшие операции, но удобство сопровождения программного кода оставляет желать лучшего, это плохой знак. Язык Python ликвидирует проблемы, связанные с сопровождением программного кода, и он действительно позволяет выражать сложные идеи простыми языковыми конструкциями. Простота программного кода – существенный фактор, который облегчает дальнейшее его сопровождение. Программный код на языке Python настолько прост, что позволяет возвращаться к нему спустя месяцы. И достаточно прост, чтобы можно было вносить изменения в программный код, который раньше нам не встречался. Таким образом, синтаксис и общие идиомы этого языка настолько ясные, краткие и простые, что позволяют работать с ним в течение длительных периодов времени.

Следующая причина, по которой мы считаем Python превосходным языком, заключается в высокой удобочитаемости программного кода. Блоки программного кода определяются по величине отступов. Отступы помогают взгляду следить за ходом выполнения программы. Кроме того, язык Python основан на «использовании слов». Под этим подразумевается, что хотя в языке Python используются свои специальные символы, основные его особенности в большинстве своем реализованы в виде ключевых слов или библиотек. Упор на слова, а не на специальные символы упрощает чтение и понимание программного кода.

Теперь, когда мы выявили некоторые преимущества языка Python, мы проведем сравнение нескольких фрагментов программного кода на языках Python, Perl и Bash. Попутно мы познакомимся еще с несколькими преимуществами языка Python. Ниже приводится простой пример на языке Bash, который выводит все возможные парные комбинации символов из набора `1`, `2` и символов из набора `a`, `b`:

```
#!/bin/bash

for a in 1 2; do
    for b in a b; do
        echo "$a $b"
    done
done
```

Вот эквивалентный фрагмент на языке Perl:

```
#!/usr/bin/perl

foreach $a ('1', '2') {
    foreach $b ('a', 'b') {
        print "$a $b\n";
    }
}
```

Это самый простой вложенный цикл. А теперь сравним эти реализации с циклом `for` в языке Python:

```
#!/usr/bin/env python

for a in [1, 2]:
    for b in ['a', 'b']:
        print a, b
```

Далее продемонстрируем использование условных инструкций в Bash, Perl и Python. Здесь используется простая условная инструкция `if/else`, с помощью которой выясняется – является ли заданный путь к файлу каталогом:

```
#!/bin/bash

if [ -d "/tmp" ] ; then
     echo "/tmp is a directory"
else
    echo "/tmp is not a directory"
fi
```

Ниже приводится эквивалентный сценарий на языке Perl:

```
#!/usr/bin/perl

if (-d "/tmp") {
    print "/tmp is a directory\n";
}
else {
```

```
    print "/tmp is not a directory\n";
}
```

А ниже – эквивалентный сценарий на языке Python:

```
#!/usr/bin/env python

import os
if os.path.isdir("/tmp"):
    print "/tmp is a directory"
else:
    print "/tmp is not a directory"
```

Еще один фактор, говорящий в пользу превосходства языка Python, – это простота поддержки объектно-ориентированного стиля программирования (ООП). А также то обстоятельство, что вас ничто не заставляет использовать ООП, если в этом нет необходимости. Но когда появляется потребность в нем, этот стиль оказывается чрезвычайно простым в применении. ООП позволяет легко и просто разделить проблему на составные функциональные части, объединенные в нечто под названием «объект». Язык Bash не поддерживает ООП, но Perl и Python поддерживают. Ниже приводится модуль на языке Perl с определением класса:

```
package Server;
use strict;

sub new {
    my $class = shift;
    my $self = {};
    $self->{IP} = shift;
    $self->{HOSTNAME} = shift;
    bless($self);
    return $self;
}

sub set_ip {
    my $self = shift;
    $self->{IP} = shift;
    return $self->{IP};
}

sub set_hostname {
    my $self = shift;
    $self->{HOSTNAME} = shift;
    return $self->{HOSTNAME};
}

sub ping {
    my $self = shift;
    my $external_ip = shift;
    my $self_ip = $self->{IP};
    my $self_host = $self->{HOSTNAME};
    print "Pinging $external_ip from $self_ip ($self_host)\n";
```

```
    return 0;
}

1;
```

И далее фрагмент, в котором он используется:

```
#!/usr/bin/perl

use Server;

$server = Server->new('192.168.1.15', 'grumbly');
$server->ping('192.168.1.20');
```

Программный код, в котором используется объектно-ориентированный модуль, достаточно прост. Однако на анализ самого модуля может потребоваться некоторое время, особенно если вы не знакомы с ООП или с особенностями реализации его поддержки в языке Perl.

Эквивалентный класс на языке Python и порядок его использования выглядят, как показано ниже:

```
#!/usr/bin/env python

class Server(object):
    def __init__(self, ip, hostname):
        self.ip = ip
        self.hostname = hostname
    def set_ip(self, ip):
        self.ip = ip
    def set_hostname(self, hostname):
        self.hostname = hostname
    def ping(self, ip_addr):
        print "Pinging %s from %s (%s)" % (ip_addr, self.ip, self.hostname)

if __name__ == '__main__':
    server = Server('192.168.1.20', 'bumbly')
    server.ping('192.168.1.15')
```

Примеры на языках Perl и Python демонстрируют некоторые из фундаментальных аспектов ООП, и вместе с тем они наглядно показывают различные особенности, которые используются в этих языках для достижения поставленной цели. Оба фрагмента решают одну и ту же задачу, но они отличаются друг от друга. Таким образом, если вы пожелаете использовать ООП, язык Python предоставит вам такую возможность. И вы достаточно легко и просто сможете включить его в свой арсенал.

Другое преимущество Python проистекает не из самого языка, а из его сообщества. В сообществе пользователей языка Python достигнуто единодушие по поводу способов решения определенных видов задач, которые вы должны (или не должны) использовать. Несмотря на то, что сам язык обеспечивает множество путей достижения одной и той же цели, соглашения, принятые в сообществе, могут рекомендовать

воздерживаться от использования некоторых из них. Например, инструкция `from module import *` в начале модуля считается вполне допустимой. Однако сообщество осуждает такое ее использование и рекомендует использовать либо инструкцию `import module`, либо инструкцию `from module import resource`. Импортирование всего содержимого модуля в пространство имен другого модуля может вызвать существенные осложнения, когда вы попытаетесь выяснить принцип действия модуля и узнать, где находятся вызываемые функции. Это конкретное соглашение поможет вам писать более понятный программный код, что позволит тем, кто будет сопровождать его, выполнять свою работу с большим удобством. Следование общепринятым соглашениям открывает вам путь к использованию наилучших приемов программирования. Мы считаем, что это идет только на пользу.

Стандартная библиотека языка Python – это еще одна замечательная особенность Python. Если применительно к языку Python вы услышите фразу «батарейки входят в комплект поставки», это лишь означает, что стандартная библиотека позволяет решать все виды задач без необходимости искать другие модули для этого. Например, несмотря на их отсутствие в самом языке, Python обеспечивает поддержку регулярных выражений, сокетов, нескольких потоков выполнения и функции для работы с датой/временем, синтаксического анализа документов XML, разбора конфигурационных файлов, функций для работы с файлами и каталогами, хранения данных, модульного тестирования, а также клиентские библиотеки для работы с протоколами http, ftp, imap, smtp и nntp и многое другое. Сразу после установки Python модули поддержки всех этих функциональных особенностей могут импортироваться вашими сценариями по мере необходимости. В вашем распоряжении имеются все перечисленные здесь функциональные возможности. Весьма впечатляет, что все это поставляется в составе Python и вам не требуется приобретать что-то еще. Все эти возможности будут чрезвычайно полезны вам при создании своих собственных программ на языке Python.

Простой доступ к огромному количеству пакетов сторонних производителей – еще одно важное преимущество Python. Помимо множества библиотек, входящих в стандартную библиотеку языка Python, существует большое число библиотек и утилит, доступных в Интернете, которые устанавливаются одной командой. В Интернете, по адресу *http://pypi.python.org*, существует каталог пакетов Python (Python Package Index, PyPI), где любой желающий может выкладывать свои пакеты в общее пользование. К моменту, когда писались эти строки, для загрузки было доступно более 3800 пакетов. В составе пакетов присутствуют IPython, который будет рассматриваться в следующей главе, Storm (модуль объектно-реляционного отображения, который будет рассматриваться в главе 12) и Twisted, сетевая платформа, которая будет рассматриваться в главе 5, – это только 3 названия из более чем

3800 пакетов. Начав пользоваться PyPI, вы обнаружите, что он совершенно необходим вам для поиска и установки полезных пакетов.

Многие из преимуществ языка Python проистекают из его базовой философии. Если в строке приглашения к вводу Python ввести команду `import this`**, перед вами появится так называемый «Дзен языка Python» Тима Петерса (Tim Peters):**

```
In [1]: import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
( Перевод:
Красивое лучше уродливого.
Явное лучше неявного.
Простое лучше сложного.
Сложное лучше усложненного.
Плоское лучше вложенного.
Разрежённое лучше плотного.
Удобочитаемость важна.
Частные случаи не настолько существенны, чтобы нарушать правила.
Однако практичность важнее чистоты.
Ошибки никогда не должны замалчиваться.
За исключением замалчивания, которое задано явно.
В случае неоднозначности сопротивляйтесь искушению угадать.
Должен существовать один, и, желательно, только один очевидный
способ сделать это.
Хотя он может быть с первого взгляда не очевиден, если ты не голландец.
Сейчас лучше, чем никогда.
Однако, никогда чаще лучше, чем *прямо* сейчас.
Если реализацию сложно объяснить, – это плохая идея.
Если реализацию легко объяснить, – это может быть хорошая идея.
Пространства имён – великолепная идея, их должно быть много!
)
```

Этот свод правил – не догма, следование которой считается обязательным на всех уровнях разработки языка, но сам дух его, кажется, пропитывает почти все, что происходит в языке и с языком. И мы считаем, что это замечательно. Возможно, именно поэтому мы день за днем стремимся использовать Python. Эта философия совпадает с тем, чего мы ждем от языка. И если она совпадает с вашими ожиданиями, значит язык Python будет хорошим выбором и для вас.

Мотивация

Если вы только что взяли эту книгу в свои руки в книжном магазине или читаете введение где-нибудь в Интернете, вы, возможно, задаете себе вопрос – насколько сложно будет изучить язык Python и стоит ли это делать. Несмотря на растущую популярность Python, многие системные администраторы до сих пор используют в своей работе только Bash и Perl. Если вы относитесь к их категории, вас наверняка обрадует тот факт, что язык Python очень прост в изучении. Хотя это вопрос личного мнения, но многие убеждены в том, что это самый простой язык для изучения и преподавания, и точка!

Если вы уже знакомы с языком Python или считаете себя гуру в программировании на другом языке, вы наверняка сможете прямо сейчас перейти к любой из следующих глав, не читая это введение, и сумеете разобраться в наших примерах. Мы старались создавать примеры, которые действительно помогут вам выполнять свою работу. В книге имеются примеры способов автоматического обнаружения и мониторинга подсетей с помощью SNMP, преобразования в интерактивную оболочку Python под названием IPython, организации конвейерной обработки данных, инструментов управления метаданными с помощью средств объектно-реляционного отображения, сетевых приложений, инструментов командной строки и многого другого.

Если у вас имеется опыт программирования на языке командной оболочки, вам тоже нет причин волноваться. Вы также легко и быстро освоите Python. Вам нужно лишь желание учиться, определенная доля любопытства и решимость – те же факторы, которые побудили вас взять в руки эту книгу и прочитать введение.

Мы понимаем, что среди вас есть и скептики. Возможно, часть из того, что вы слышали о программировании, напугала вас. Существует одно общее, глубоко неверное заблуждение, что программированию могут научиться не все, а только избранная таинственная элита. На самом деле любой желающий может научиться программировать. Второе, не менее ложное заблуждение состоит в том, что только получение профессионального образования в области информатики открывает человеку путь к званию программиста. Однако у некоторых самых талантливых программистов нет диплома об образовании в данной области. Среди компетентных программистов на языке Python имеются люди

с профессиональной подготовкой в области философии, журналистики, диетологии и английского языка. Наличие специального образования не является обязательным требованием для освоения Python, хотя оно и не повредит.

Еще одно забавное и такое же неверное представление заключается в том, что программированию можно учиться только в подростковом возрасте. Хотя это позволяет хорошо себя чувствовать людям, которым посчастливилось встретить в юности кого-то, кто вдохновил их заняться программированием, тем не менее, это еще один миф. Очень полезно начинать изучение программирования в юном возрасте, но возраст не является препятствием к освоению языка Python. Освоение Python – это не «игрушка для молодежи», как иногда говорят. Среди разработчиков существует бесчисленное множество людей, которые осваивали программирование, будучи в возрасте старше 20, 30, 40 лет и даже больше.

Если вы добрались до этого места, то надо заметить, что у вас, уважаемый читатель, есть одно преимущество, которое отсутствует у многих. Если вы решили взять в руки книгу, рассказывающую об использовании языка Python в системном администрировании UNIX и Linux, значит, вы уже представляете себе, как выполнять команды в командной оболочке. Это огромное преимущество для того, кто решил стать программистом на языке Python. Наличие знаний о способах выполнения команд в терминале – это все, что необходимо для этого введения в Python. Если вы уверены, что научитесь программированию на языке Python, тогда сразу же переходите к следующему разделу. Если у вас еще есть сомнения, тогда прочитайте этот раздел еще раз и убедите себя в том, что вы в силах овладеть программированием на языке Python. Это действительно просто, и если вы примете такое решение, оно изменит вашу жизнь.

Основы

Это введение в язык Python сильно отличается от любого другого, т. к. мы будем использовать интерактивную оболочку под названием IPython и обычную командную оболочку Bash. Вы должны будете открыть два окна терминалов, одно – с командной оболочкой Bash и другое – с интерактивной оболочкой IPython. В каждом примере мы будем сравнивать, как выполняются одни и те же действия с помощью Python и с помощью Bash. Для начала загрузите требуемую версию интерактивной оболочки IPython для своей платформы и установите ее. Получить ее можно на странице *http://ipython.scipy.org/moin/Download*. Если по каким-то причинам вы не можете получить и установить IPython, то можно использовать обычную интерактивную оболочку интерпретатора Python. Вы можете также загрузить копию виртуальной машины, включающую в себя все примеры программ из книги, а также предварительно настроенную и готовую к работе интер-

активную оболочку IPython. Вам достаточно просто ввести команду `ipython`, и вы попадете в строку приглашения к вводу.

Как только оболочка IPython будет установлена и запущена, вы должны увидеть примерно следующее:

```
[ngift@Macintosh-7][H:10679][J:0]# ipython
Python 2.5.1 (r251:54863, Jan 17 2008, 19:35:17)
Type "copyright", "credits" or "license" for more information.

IPython 0.8.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]:
```

Интерактивная оболочка IPython напоминает обычную командную оболочку Bash и может выполнять такие команды, как `ls`, `cd` и `pwd`, а в следующей главе приведены более подробные сведения о IPython. Эта глава посвящена изучению Python.

В окне терминала с интерактивной оболочкой Python введите следующую команду:

```
In [1]: print "I can program in Python"
I can program in Python
```

В окне терминала с командной оболочкой Bash введите следующую команду:

```
[ngift@Macintosh-7][H:10688][J:0]# echo "I can program in Bash"
I can program in Bash
```

В этих двух примерах не ощущается существенных различий между Python и Bash. Мы надеемся, что это поможет лишить Python части его загадочности.

Выполнение инструкций в языке Python

Если вам приходится тратить значительную часть времени на ввод команд в окне терминала, значит, вам уже приходилось выполнять инструкции и, возможно, перенаправлять вывод в файл или на вход другой команды UNIX. Рассмотрим порядок выполнения команды в Bash, а затем сравним его с порядком, принятым в Python. В окне терминала с командной оболочкой Bash введите следующую команду:

```
[ngift@Macintosh-7][H:10701][J:0]# ls -l /tmp/
total 0
-rw-r--r--  1 ngift  wheel  0 Apr  7 00:26 file.txt
```

В окне терминала с интерактивной оболочкой Python введите следующую команду:

```
In [2]: import subprocess

In [3]: subprocess.call(["ls","-l ","/tmp/"])
total 0
-rw-r--r--  1 ngift  wheel  0 Apr  7 00:26 file.txt
Out[3]: 0
```

Пример с командной оболочкой Bash не нуждается в пояснениях, так как это обычная команда `ls`, но, если ранее вам никогда не приходилось сталкиваться с программным кодом на языке Python, пример с интерактивной оболочкой Python наверняка покажется вам немного странным. Вы могли бы подумать: «Это еще что за команда `import subprocess`?». Одна из самых важных особенностей Python заключается в его возможности импортировать модули или другие файлы, содержащие программный код и используемые в новых программах. Если вам знакома инструкция `source` в Bash, то вы увидите определенное сходство с ней. В данном конкретном случае важно понять, что вы импортировли модуль subprocess и использовали его посредством синтаксической конструкции, показанной выше. Подробнее о том, как работают `subprocess` и `import`, мы расскажем позже, а пока не будем задумываться о том, почему эта инструкция работает, и обратимся к следующей строке:

```
subprocess.call(["команда", "аргумент", "другой_аргумент_или_путь"])
```

Из Python можно выполнить любую команду оболочки, как если бы она выполнялась командной оболочкой Bash. Учитывая эту информацию, можно сконструировать версию команды `ls` на языке Python. Откройте предпочтительный текстовый редактор или новую вкладку в окне терминала, создайте файл с именем *pyls.py* и сделайте его выполняемым с помощью команды `chmod +x pyls.py`. Содержимое файла приводится в примере 1.1.

Пример 1.1. Версия команды ls на языке Python

```
#!/usr/bin/env python
#Версия команды ls на языке Python

import subprocess

subprocess.call(["ls","-l"])
```

Если теперь запустить этот сценарий, вы получите тот же самый результат, что и при запуске команды `ls -l` из командной строки:

```
[ngift@Macintosh-7][H:10746][J:0]# ./pyls.py
total 8
-rwxr-xr-x  1 ngift  staff  115 Apr  7 12:57 pyls.py
```

Этот пример может показаться глупым (да он таким и является), но он наглядно демонстрирует типичное применение Python в системном администрировании. Язык Python часто используется для «обертывания» других сценариев или команд UNIX. Теперь вы уже смогли бы

начать писать некоторые несложные сценарии, помещая в файл команды одну за другой. Рассмотрим простые примеры, которые реализованы именно таким способом. Вы можете либо просто скопировать содержимое примера 1.2, либо выполнить сценарии *psysinfo.py* и *bashsysinfo.py*, которые можно найти в примерах к этой главе.

Пример 1.2. Сценарий получения информации о системе – Python

```
#!/usr/bin/env python
#Сценарий сбора информации о системе
import subprocess

#Команда 1
uname = "uname"
uname_arg = "-a"
print "Gathering system information with %s command:\n" % uname
subprocess.call([uname, uname_arg])

#Команда 2
diskspace = "df"
diskspace_arg = "-h"
print "Gathering diskspace information %s command:\n" % diskspace
subprocess.call([diskspace, diskspace_arg])
```

Пример 1.3. Сценарий получения информации о системе – Bash

```
#!/usr/bin/env bash
#Сценарий сбора информации о системе

#Команда 1
UNAME="uname -a"
printf "Gathering system information with the $UNAME command: \n\n"
$UNAME

#Команда 2
DISKSPACE="df -h"
printf "Gathering diskspace information with the $DISKSPACE command: \n\n"
$DISKSPACE
```

Если внимательно рассмотреть оба сценария, можно заметить, что они очень похожи. А если запустить их, будут получены идентичные результаты. Маленькое примечание: передавать в функции `subprocess.call` команду отдельно от аргумента совершенно необязательно. Можно использовать, например, такую форму записи:

```
subprocess.call("df -h", shell=True)
```

Все замечательно, но мы еще не объяснили, как действует инструкция `import` и что из себя представляет модуль `subprocess`. В версии сценария на языке Python мы выполнили импортирование модуля `subprocess`, т. к. он содержит программный код, позволяющий выполнять вызов команд системы.

Как уже упоминалось ранее, импортируя модуль `subprocess`, мы просто импортируем файл, содержащий необходимый нам программный код. Вы можете создать свой собственный модуль, или файл, и неоднократно использовать написанный вами программный код, импортируя его точно так же, как мы импортировали программный код из модуля `subprocess`. В импортировании нет ничего необычного, просто в результате этой операции вы получаете в свое распоряжение файл с некоторым программным кодом в нем. Одна из замечательных особенностей интерактивной оболочки IPython состоит в ее способности заглядывать внутрь модулей и файлов и получать списки доступных атрибутов. Если говорить терминами UNIX, это напоминает действие команды `ls` в каталоге */usr/bin*. Например, если вы оказались в новой системе, такой как Ubuntu или Solaris, а привыкли работать с Red Hat, то вы можете выполнить команду `ls` в каталоге */usr/bin*, чтобы узнать – имеется ли в наличии такой инструмент, как wget, curl или lynx. Если вы хотите воспользоваться инструментом, находящимся в каталоге */usr/bin*, можно просто ввести команду `/usr/bin/wget`, например.

Ситуация с модулями, такими как `subprocess`, очень похожа на описанную выше. В интерактивной оболочке IPython можно использовать функцию автодополнения, чтобы увидеть, какие инструменты доступны внутри модуля. Воспользуемся возможностью автодополнения и посмотрим, какие атрибуты имеются внутри модуля `subprocess`. Не забывайте, что модуль – это всего лишь файл с некоторым программным кодом внутри него. Ниже показано, что возвращает функция автодополнения в IPython для модуля `subprocess`:

```
In [12]: subprocess.
subprocess.CalledProcessError subprocess.__hash__      subprocess.call
subprocess.MAXFD              subprocess.__init__      subprocess.check_call
subprocess.PIPE               subprocess.__name__      subprocess.errno
subprocess.Popen              subprocess.__new__       subprocess.fcntl
subprocess.STDOUT             subprocess.__reduce__    subprocess.list2cmdline
subprocess.__all__            subprocess.__reduce_ex__ subprocess.mswindows
subprocess.__builtins__       subprocess.__repr__      subprocess.os
subprocess.__class__          subprocess.__setattr__   subprocess.pickle
subprocess.__delattr__        subprocess.__str__       subprocess.select
subprocess.__dict__           subprocess._active       subprocess.sys
subprocess.__doc__            subprocess._cleanup      subprocess.traceback
subprocess.__file__           subprocess._demo_posix   subprocess.types
subprocess.__getattribute__   subprocess._demo_windows
```

Чтобы воспроизвести этот эффект, вам нужно просто ввести команду:

```
import subprocess
```

затем ввести:

```
subprocess.
```

и нажать клавишу Tab, чтобы активизировать функцию автодополнения, которая выведет список доступных атрибутов. В третьей колонке нашего примера можно заметить `subprocess.call`. Теперь, чтобы получить дополнительную информацию об использовании `subprocess.call`, введите команду:

```
In [13]: subprocess.call?

Type:        function
Base Class:  <type 'function'>
String Form: <function call at 0x561370>
Namespace:   Interactive
File: /System/Library/Frameworks/Python.framework/Versions/2.5/lib/
python2.5/
      subprocess.py
Definition:   subprocess.call(*popenargs, **kwargs)
Docstring:
    Run command with arguments. Wait for command to complete, then
    return the returncode attribute.
    (Запускает команду с аргументами. Ожидает ее завершения и возвращает
    атрибут returncode)

    The arguments are the same as for the Popen constructor. Example:
    (Аргументы те же, что и в конструкторе Popen. Например:)
    retcode = call(["ls", "-l"])
```

Символ вопросительного знака в данном случае трактуется как обращение к странице справочного руководства. Когда требуется узнать, как работает некоторый инструмент в системе UNIX, достаточно ввести команду:

```
man имя_инструмента
```

То же и с атрибутом внутри модуля, таким как `subprocess.call`. Когда в оболочке IPython после имени атрибута вводится вопросительный знак, выводится документация, которая была включена в атрибут. Если подобную операцию выполнить с атрибутами из стандартной библиотеки, вы сможете обнаружить достаточно полезную информацию по их использованию. Имейте в виду, что существует также возможность обратиться к документации с описанием стандартной библиотеки языка Python.

Когда мы смотрим на это описание, в стандартный раздел «Docstring», мы видим пример использования атрибута `subprocess.call` и описание того, что он делает.

Итог

Теперь вы обладаете объемом знаний, достаточным, чтобы называть себя программистом на языке Python. Вы знаете, как написать простейший сценарий на языке Python, как перевести сценарий с языка Bash на язык Python, и наконец, вы знаете, как отыскать описание мо-

дулей и атрибутов. В следующем разделе вы узнаете, как организовать эти простые последовательности команд в функции.

Использование функций в языке Python

В предыдущем разделе мы узнали, как выполняются инструкции, что само по себе весьма полезно, т. к. это означает, что мы в состоянии автоматизировать выполнение некоторых операций, которые раньше выполнялись вручную. Следующим шагом к нашему программному коду автоматизации будет создание функций. Если вы еще не знакомы с функциями в языке Bash или в каком-либо другом языке программирования, то просто представляйте их себе как мини-сценарии. Функции позволяют создавать блоки инструкций, которые работают в группе. Это немного похоже на сценарий Bash с двумя командами, написанный нами ранее. Одно из отличий состоит в том, что вы можете включить в сценарий множество функций. В конечном счете можно весь программный код сценария расположить в функциях и затем запускать эти мини-программы в нужное время в своем сценарии.

Теперь настало время поговорить об отступах. В языке Python строки, принадлежащие одному и тому же блоку программного кода, должны иметь одинаковые отступы. В других языках, таких как Bash, когда определяется функция, ее тело заключается в фигурные скобки. В языке Python все строки в скобках должны иметь одинаковые отступы. Это может сбивать с толку тех, кто только начинает изучать язык, но через некоторое время это войдет в привычку и вы заметите, что выполнение этого требования повышает удобочитаемость программного кода. Если при работе с какими-либо примерами из книги у вас появляются ошибки, убедитесь для начала, что в исходных текстах правильно соблюдены отступы. Обычно один шаг отступа принимают равным четырем пробелам.

Рассмотрим, как работают функции в языке Python и Bash. Если у вас по-прежнему открыта интерактивная оболочка IPython, вы можете не создавать файл сценария на языке Python, хотя это и не возбраняется. Просто введите следующий текст в строке приглашения оболочки IPython:

```
In [1]: def pyfunc():
...:     print "Hello function"
...:
...:

In [2]: pyfunc
Out[2]: <function pyfunc at 0x2d5070>

In [3]: pyfunc()
Hello function

In [4]: for i in range(5):
```

```
...:     pyfunc()
...:
...:
Hello function
Hello function
Hello function
Hello function
Hello function
```

В этом примере инструкция `print` помещена в функцию. Теперь можно не только вызвать эту функцию позднее, но и вызвать ее столько раз, сколько потребуется. В строке `[4]` была использована идиома (прием) для выполнения функции пять раз. Если раньше вам такой прием не встречался, постарайтесь понять, что он вызывает функцию пять раз.

То же самое можно сделать непосредственно в командной оболочке Bash. Ниже демонстрируется один из способов:

```
bash-3.2$ function shfunc()
> {
>     printf "Hello function\n"
> }
bash-3.2$ for (( i=0 ; i < 5 ; i++))
> do
>     shfunc
> done
Hello function
Hello function
Hello function
Hello function
Hello function
```

В примере на языке Bash была создана простая функции `shfunc`, которая затем была вызвана пять раз, точно так же, как это было сделано ранее с функцией в примере на языке Python. Примечательно, что в примере на языке Bash потребовалось больше «багажа», чтобы реализовать то же самое, что и на языке Python. Обратите внимание на отличия цикла `for` в языке Bash от цикла `for` в языке Python. Если это ваша первая встреча с функциями в Bash или Python, вам следует поупражняться в создании каких-нибудь других функций в окне IPython, прежде чем двигаться дальше.

В функциях нет ничего таинственного, и попытка написать несколько функций в интерактивной оболочке поможет ликвидировать налет таинственности в случае, если это ваш первый опыт работы с функциями. Ниже приводится пара примеров простых функций:

```
In [1]: def print_many():
...:     print "Hello function"
...:     print "Hi again function"
...:     print "Sick of me yet"
...:
```

```
...:
In [2]: print_many()
Hello function
Hi again function
Sick of me yet

In [3]: def addition():
...:     sum = 1+1
...:     print "1 + 1 = %s" % sum
...:
...:

In [4]: addition()
1 + 1 = 2
```

Итак, у нас за плечами имеется несколько простейших примеров кроме тех, что вы попробовали выполнить сами, не правда ли? Теперь мы можем вернуться к сценарию, который собирает информацию о системе и реализовать его с применением функций, как показано в примере 1.4.

Пример 1.4. Преобразованный сценарий сбора информации о системе на языке Python: pysysinfo_func.py

```
#!/usr/bin/env python
#Сценарий сбора информации о системе
import subprocess

#Команда 1
def uname_func():

    uname = "uname"
    uname_arg = "-a"
    print "Gathering system information with %s command:\n" % uname
    subprocess.call([uname, uname_arg])

#Команда 2
def disk_func():

    diskspace = "df"
    diskspace_arg = "-h"
    print "Gathering diskspace information %s command:\n" % diskspace
    subprocess.call([diskspace, diskspace_arg])

#Главная функция, которая вызывает остальные функции
def main():
    uname_func()
    disk_func()

main()
```

Учитывая наши эксперименты с функциями, можно сказать, что преобразование предыдущей версии сценария вылилось в то, что мы просто поместили инструкции внутрь функций и затем организовали их вызов с помощью главной функции. Если вы не знакомы с подобным

стилем программирования, тогда, возможно, вы не знаете, что это достаточно распространенный прием, когда внутри сценария создается несколько функций, а затем они вызываются из одной главной функции. Одна из множества причин для такой организации состоит в том, что, когда вы решите использовать этот сценарий с другой программой, вы сможете выбирать, вызывать ли функции по отдельности или с помощью главной функции. Суть в том, что решение принимается после того, как модуль будет импортирован.

Когда нет никакого управления потоком выполнения или главной функции, весь программный код выполняется немедленно, во время импортирования модуля. Это может быть и неплохо для одноразовых сценариев, но если вы предполагаете создавать инструменты многократного пользования, тогда лучше будет использовать функции, которые заключают в себе определенные действия, и предусматривать создание главной функции, которая будет выполнять всю программу целиком.

Для сравнения также используем функции для предыдущего сценария на языке Bash, выполняющего сбор информации о системе, как показано в примере 1.5.

Пример 1.5. Преобразованный сценарий сбора информации о системе на языке Bash: bashsysinfo_func.sh

```
#!/usr/bin/env bash
#Сценарий сбора информации о системе

#Команда 1
function uname_func ()
{
    UNAME="uname -a"
    printf "Gathering system information with the $UNAME command: \n\n"
    $UNAME
}

#Команда 2
function disk_func ()
{
    DISKSPACE="df -h"
    printf "Gathering diskspace information with the $DISKSPACE command:
\n\n"
    $DISKSPACE
}

function main ()
{
    uname_func
    disk_func
}

main
```

Взглянув на наш пример на языке Bash, можно заметить немало схожего с аналогичным ему сценарием на языке Python. Здесь также созданы две функции, которые затем вызываются из главной функции. Если это ваш первый опыт работы с функциями, то мы могли бы порекомендовать вам закомментировать вызов главной функции в обоих сценариях, поставив в начале строки символ решетки (#), и попробовать запустить их еще раз. На этот раз в результате запуска сценариев вы не должны получить ровным счетом ничего, потому что программа хотя и выполняется, но она не вызывает две свои функции.

Теперь вы можете считать себя программистом, способным писать простые функции на обоих языках, Bash и Python. Программисты учатся работая, поэтому сейчас мы настоятельно рекомендуем вам изменить в обеих программах, на языке Bash и Python, вызовы системных команд своими собственными. Прибавьте себе несколько очков, если вы добавили в сценарии несколько новых функций и предусмотрели их вызов из главной функции.

Повторное использование программного кода с помощью инструкции import

Одна из проблем с освоением чего-либо нового состоит в том, что если это новое достаточно абстрактная вещь, бывает очень сложно найти ей применение. Когда в последний раз вам приходилось применять знание математики, полученное в средней школе, в продуктовом магазине? В предыдущих примерах было показано, как создавать функции, которые представляют альтернативу простому последовательному выполнению команд оболочки. Мы также сообщили, что модуль – это обычный сценарий или некоторое количество строк программного кода в файле. В этом подходе нет ничего сложного, но программный код должен быть организован определенным способом, чтобы его можно было повторно использовать в будущих программах. В этом разделе мы покажем вам, почему это так важно. Давайте импортируем оба предыдущих сценария сбора информации о системе и выполним их.

Откройте окна с IPython и Bash, если вы закрыли их, чтобы мы могли быстро продемонстрировать, почему функции играют такую важную роль с точки зрения повторного использования программного кода. Один из наших первых сценариев на языке Python представлял собой простую последовательность команд в файле с именем *pysysinfo.py*. В языке Python файл является модулем и наоборот, поэтому мы можем импортировать этот файл сценария в оболочку IPython. Обратите внимание, вы никогда не должны указывать расширение *.py* файла в инструкции импорта. Фактически попытка импорта окончится неудачей, если расширение будет указано. Итак, мы выполнили импорт сценария на ноутбуке Hoa Macbook Pro:

```
In [1]: import pysysinfo
Gathering system information with uname command:

Darwin Macintosh-8.local 9.2.2 Darwin Kernel Version 9.2.2: /
  Tue Mar 4 21:17:34 PST 2008; root:xnu-1228.4.31~1/RELEASE_I386 i386
Gathering diskspace information df command:

Filesystem     Size   Used  Avail Capacity Mounted on
/dev/disk0s2   93Gi   88Gi  4.2Gi    96%   /
devfs         110Ki  110Ki    0Bi   100%   /dev
fdesc         1.0Ki  1.0Ki    0Bi   100%   /dev
map -hosts      0Bi    0Bi    0Bi   100%   /net
map auto_home   0Bi    0Bi    0Bi   100%   /home
/dev/disk1s2  298Gi  105Gi  193Gi    36%   /Volumes/Backup
/dev/disk2s3  466Gi  240Gi  225Gi    52%   /Volumes/EditingDrive
```

Ух ты! Выглядит круто, правда? Когда импортируется файл, содержащий программный код на языке Python, он тут же выполняется. Но в действительности за всем этим кроется несколько проблем. Если вы планируете запускать такой программный код на языке Python, его всегда придется запускать из командной строки как часть сценария или программы, которую вы пишете. Операция импорта должна помочь в воплощении идеи «повторного использования программного кода». Но вот что интересно: как быть, если нам потребуется получить только информацию о распределении дискового пространства? В данном сценарии это невозможно. Именно для этого используются функции. Они позволяют контролировать, когда и какие части программы должны выполняться, чтобы она не выполнялась целиком, как в примере выше. Если импортировать сценарий, где эти команды оформлены в виде функций, можно увидеть, что мы имеем в виду.

Ниже приводится результат импортирования сценария в терминале IPython:

```
In [3]: import pysysinfo_func
Gathering system information with uname command:

Darwin Macintosh-8.local 9.2.2 Darwin Kernel Version 9.2.2: /
  Tue Mar 4 21:17:34 PST 2008; root:xnu-1228.4.31~1/RELEASE_I386 i386
Gathering diskspace information df command:

Filesystem     Size   Used  Avail Capacity Mounted on
/dev/disk0s2   93Gi   88Gi  4.2Gi    96%   /
devfs         110Ki  110Ki    0Bi   100%   /dev
fdesc         1.0Ki  1.0Ki    0Bi   100%   /dev
map -hosts      0Bi    0Bi    0Bi   100%   /net
map auto_home   0Bi    0Bi    0Bi   100%   /home
/dev/disk1s2  298Gi  105Gi  193Gi    36%   /Volumes/Backup
/dev/disk2s3  466Gi  240Gi  225Gi    52%   /Volumes/EditingDrive
```

Этот результат ничем не отличается от того, что был получен при использовании сценария без функций. Если вы озадачены, – это хороший знак. Чтобы понять, почему был получен тот же самый результат, дос-

таточно заглянуть в исходный программный код. Откройте сценарий *pysysinfo_func.py* в другой вкладке или в другом окне терминала и найдите строки:

```
#Главная функция, которая вызывает остальные функции
def main():
    uname_func()
    disk_func()

main()
```

Проблема в том, что функция `main`, созданная нами в конце предыдущего раздела, обернулась для нас некоторой неприятностью. С одной стороны, хотелось бы иметь возможность запускать сценарий из командной строки, чтобы получать полную информацию о системе, но с другой стороны, нам совсем не нужно, чтобы модуль выводил что-либо при импортировании. К счастью, потребность использовать модули как в виде сценариев, выполняемых из командной строки, так и в виде повторно используемых модулей достаточно часто встречается в языке Python. Решение этой проблемы состоит в том, чтобы определить, когда следует вызывать главную функцию, изменив последнюю часть сценария, как показано ниже:

```
#Главная функция, которая вызывает остальные функции
def main():
    uname_func()
    disk_func()

if __name__ == "__main__":
    main()
```

Эта «идиома» представляет прием, который обычно используется для решения данной проблемы. Любой программный код, входящий в состав блока этой условной инструкции, будет выполняться, только когда модуль запускается из командной строки. Чтобы убедиться в этом, измените окончание своего сценария или импортируйте исправленную его версию *pysysinfo_func_2.py*.

Если теперь вернуться к оболочке IPython и импортировать новый сценарий, вы должны увидеть следующее:

```
In [1]: import pysysinfo_func_2
```

На этот раз благодаря нашим исправлениям функция `main` вызвана не была. Итак, вернемся вновь к теме повторного использования программного кода: у нас имеется три функции, которые можно использовать в других программах или вызывать в интерактивной оболочке IPython. Вспомните: ранее мы говорили, что было бы неплохо иметь возможность вызвать только функцию, которая выводит информацию о распределении дискового пространства. Сначала необходимо вновь вернуться к одной из возможностей оболочки IPython, которую мы уже демонстрировали ранее. Вспомните, как мы использовали клавишу Tab