Python для сетевых инженеров

Выпуск 3.0

апр. 16, 2021

Оглавление

1 Введение **3**

О книге . . . 3
Для кого эта книга . . . 3
Зачем Вам учиться программировать? . . . 3
Требуемые версии ОС и Python . . . 4
Примеры . . . 4
Задания . . . 4
Вопросы . . . 5
Презентации . . . 5
Форматы файлов книги . . . 6
Обсуждение . . . 6
Часто задаваемые вопросы (FAQ) . . . 6
Будет ли печатная версия книги? . . . 6
Почему в книге нет темы X? . . . 6
Чем это отличается от обычного вводного курса по Python? . . . 6
Я сетевик. Для чего мне нужна эта книга? . . . 6
Почему книга именно для сетевых инженеров? . . . 7
Почему именно Python? . . . 8
Нужный мне модуль не поддерживает Python 3 . . . 8
Я не знаю, нужно ли мне это. . . . 9
Зачем сетевому инженеру программирование? . . . 9
Книга будет когда-то платной? . . . 10
Благодарности . . . 10

2 Ресурсы для обучения по книге **11**

Подготовка рабочего окружения . . . 11
Задания . . . 11
Тесты . . . 12
Дополнительные материалы . . . 13

3 I. Основы Python **15**
1. Подготовка к работе . . . 16
Подготовка рабочего окружения . . . 17
ОС и редактор . . . 20
Система управления пакетами pip . . . 21
Виртуальные окружения . . . 22
Интерпретатор Python . . . 26
Дополнительные материалы . . . 26
Задания . . . 28
2. Использование Git и GitHub . . . 29
Основы Git . . . 29
Отображение статуса репозитория в приглашении . . . 30
Работа с Git . . . 31
Дополнительные возможности . . . 35
Аутентификация на GitHub . . . 39
Работа со своим репозиторием заданий . . . 40
Работа с репозиторием заданий и примеров . . . 43
Дополнительные материалы . . . 46
Задания . . . 47
3. Начало работы с Python . . . 48
Синтаксис Python . . . 48
Интерпретатор Python. IPython . . . 50
Специальные команды ipython . . . 55
Переменные . . . 58
Задания . . . 61
4. Типы данных в Python . . . 62
Числа . . . 62
Строки (Strings) . . . 65
Список (List) . . . 79
Словарь (Dictionary) . . . 84
Кортеж (Tuple) . . . 93
Множество (Set) . . . 95
Булевы значения . . . 97
Преобразование типов . . . 98
Проверка типов . . . 100
Вызов методов цепочкой . . . 102
Основы сортировки данных . . . 104
Дополнительные материалы . . . 104
Задания . . . 106
5. Создание базовых скриптов . . . 110
Исполняемый файл . . . 110
Передача аргументов скрипту (argv) . . . 111
Ввод информации пользователем . . . 112
Задания . . . 114
6. Контроль хода программы . . . 123

if/elif/else . . . 123
for . . . 129
while . . . 136
break, continue, pass . . . 138
for/else, while/else . . . 141
Работа с исключениями try/except/else/finally . . . 143
Дополнительные материалы . . . 149
Задания . . . 151
7. Работа с файлами . . . 154
Открытие файлов . . . 154
Чтение файлов . . . 155
Запись файлов . . . 160
Закрытие файлов . . . 163
Конструкция with . . . 164
Примеры работы с файлами . . . 167
Дополнительные материалы . . . 173
Задания . . . 174
8. Примеры использования основ . . . 178
Форматирование строк с помощью f-строк . . . 178
Распаковка переменных . . . 183
List, dict, set comprehensions . . . 189
Дополнительные материалы . . . 197

4 II. Повторное использование кода 199

9. Функции . . . 200
Создание функций . . . 200
Пространства имен. Области видимости . . . 205
Параметры и аргументы функций . . . 206
Аргументы, которые можно передавать только как ключевые . . . 221
Распространенные проблемы/нюансы работы с функциями . . . 222
Дополнительные материалы . . . 225
Задания . . . 226
10. Полезные функции . . . 233
Функция print . . . 233
Функция range . . . 236
Функция sorted . . . 239
enumerate . . . 243
Функция zip . . . 245
Функция all . . . 248
Функция any . . . 249
Анонимная функция (лямбда-выражение) . . . 250
Функция map . . . 251
Функция filter . . . 253
11. Модули . . . 255
Импорт модуля . . . 255

Создание своих модулей . . . 258
if __name__ == "__main__" . . . 260
Пути поиска модулей . . . 262
Рекомендации по поводу расположения функций в коде . . . 264
Задания . . . 265
12. Полезные модули . . . 270
Модуль subprocess . . . 270
Модуль os . . . 275
Модуль ipaddress . . . 277
Модуль tabulate . . . 282
Модуль pprint . . . 286
Дополнительные материалы . . . 291
Задания . . . 292
13. Итераторы, итерируемые объекты и генераторы . . . 294
Итерируемый объект . . . 294
Итераторы . . . 295
Генератор (generator) . . . 297
Дополнительные материалы . . . 299

5 III. Регулярные выражения 301
14. Синтаксис регулярных выражений . . . 302
Синтаксис регулярных выражений . . . 302
Наборы символов . . . 304
Символы повторения . . . 305
Специальные символы . . . 310
Жадность символов повторения . . . 315
Группировка выражений . . . 316
Разбор вывода команды show ip dhcp snooping с помощью именованных групп . . . 318
Группа без захвата . . . 321
Повторение захваченного результата . . . 322
15. Модуль re . . . 324
Объект Match . . . 324
Функция search . . . 331
Функция match . . . 336
Функция finditer . . . 338
Функция findall . . . 342
Функция compile . . . 344
Флаги . . . 348
Функция re.split . . . 351
Функция re.sub . . . 353
Дополнительные материалы . . . 354
Задания . . . 356

6 IV. Запись и передача данных 361
16. Unicode . . . 363

Стандарт Юникод . . . 363
Юникод в Python 3 . . . 364
Конвертация между байтами и строками . . . 367
Примеры конвертации между байтами и строками . . . 368
Ошибки при конвертации . . . 372
Дополнительные материалы . . . 375
17. Работа с файлами в формате CSV, JSON, YAML . . . 376
Работа с файлами в формате CSV . . . 376
Работа с файлами в формате JSON . . . 382
Работа с файлами в формате YAML . . . 389
Дополнительные материалы . . . 394
Задания . . . 395

7 V. Работа с сетевым оборудованием 401
18. Подключение к оборудованию . . . 402
Ввод пароля . . . 403
Модуль pexpect . . . 404
Модуль telnetlib . . . 413
Модуль paramiko . . . 422
Модуль netmiko . . . 427
Модуль scrapli . . . 435
Дополнительные материалы . . . 447
Задания . . . 449
19. Одновременное подключение к нескольким устройствам . . . 457
Измерение времени выполнения скрипта . . . 457
Процессы и потоки в Python (CPython) . . . 458
Количество потоков . . . 460
Потоковая безопасность . . . 461
Модуль logging . . . 463
Модуль concurrent.futures . . . 465
Дополнительные материалы . . . 482
Задания . . . 484
20. Шаблоны конфигураций с Jinja2 . . . 491
Начало работы с Jinja2 . . . 491
Пример использования Jinja . . . 493
Синтаксис шаблонов Jinja2 . . . 495
Наследование шаблонов . . . 520
Дополнительные материалы . . . 525
Задания . . . 526
21. Обработка вывода команд TextFSM . . . 531
Начало работы с TextFSM . . . 531
Синтаксис шаблонов TextFSM . . . 533
Правила состояний . . . 536
Примеры использования TextFSM . . . 538
TextFSM CLI Table . . . 555

Дополнительные материалы . . . 560
Задания . . . 561

8 VI. Основы объектно-ориентированного программирования 565
22. Основы ООП . . . 566
Основы ООП . . . 566
Создание класса . . . 568
Создание метода . . . 569
Параметр self . . . 571
Метод `__init__` . . . 573
Пример класса . . . 575
Область видимости . . . 576
Переменные класса . . . 576
Задания . . . 579
23. Специальные методы . . . 589
Подчеркивание в именах . . . 589
Методы __str__, __repr__ . . . 593
Поддержка арифметических операторов . . . 595
Протоколы . . . 598
Задания . . . 612
24. Наследование . . . 617
Основы наследования . . . 617
Задания . . . 622

9 VII. Работа с базами данных 629
25. Работа с базами данных . . . 630
SQL . . . 630
SQLite . . . 631
Основы SQL (в sqlite3 CLI) . . . 633
Модуль sqlite3 . . . 653
Дополнительные материалы . . . 679
Задания . . . 680

10 VIII. Дополнительная информация 695
Модуль argparse . . . 695
Вложенные парсеры . . . 700
Форматирование строк с оператором `%` . . . 707
Соглашение об именах . . . 708
Имена переменных . . . 709
Имена модулей и пакетов . . . 709
Имена функций . . . 709
Имена классов . . . 709
Подчеркивание в именах . . . 710
Подчеркивание как имя . . . 710
Два подчеркивания . . . 712
Два подчеркивания перед именем . . . 712

Два подчеркивания перед и после имени . . . 713
Отличия Python 2.7 и Python 3.6 . . . 715
Unicode . . . 715
Функция print . . . 715
input вместо raw_input . . . 716
range вместо xrange . . . 716
Методы словарей . . . 717
Распаковка переменных . . . 718
Итератор вместо списка . . . 718
subprocess.run . . . 719
Jinja2 . . . 719
Модули pexpect, telnetlib, paramiko . . . 719
Мелочи . . . 719
Дополнительная информация . . . 720
Проверка заданий с помощью утилиты pyneng . . . 721
Где решать задания . . . 721
Установка скрипта pyneng . . . 721
Скрипт pyneng . . . 722
Проверка заданий тестами . . . 722
Получение ответов . . . 723
Вывод pyneng . . . 723
Проверка заданий с помощью pytest . . . 726
Основы pytest . . . 726
Особенности использования pytest для проверки заданий . . . 731
pytest-clarity . . . 736

11 Продолжение обучения 739
Написание скриптов для автоматизации рабочих процессов . . . 739
Python для автоматизации работы с сетевым оборудованием . . . 740
Python без привязки к сетевому оборудованию . . . 741
Книги . . . 741
Курсы . . . 742
Сайты с задачами . . . 742
Подкасты . . . 742
Документация . . . 742

12 Отзывы читателей книги и слушателей курса 743
Ян Коробов . . . 743
Сергей Забабурин . . . 744
Александр Романов . . . 745
Денис Карпушин . . . 745
Евгений Овчинников . . . 745
Олег Востриков . . . 747
Эмиль Гарипов . . . 747
Илья про книгу . . . 748

Алексей Кириллов про онлайн курс 748
Слава Скороход про онлайн курс 749
Марат Сибгатулин про онлайн курс 750
Кирилл Плетнёв про книгу 750
Алексей про книгу 751

13 Скачать PDF/Epub **753**

В книге рассматриваются основы Python с примерами и заданиями построенными на сетевой тематике.

С одной стороны, книга достаточно базовая, чтобы её мог одолеть любой желающий, а с другой стороны, в книге рассматриваются все основные темы, которые позволят дальше расти самостоятельно. Книга не ставит своей целью глубокое рассмотрение Python. Задача книги – объяснить понятным языком основы Python и дать понимание необходимых инструментов для его практического использования. Всё, что рассматривается в книге, ориентировано на сетевое оборудование и работу с ним. Это даёт возможность сразу использовать в работе сетевого инженера то, что было изучено на курсе. Все примеры показываются на примере оборудования Cisco, но, конечно же, они применимы и для любого другого оборудования.

Примечание: В книге используется Python 3.7.

При желании, вы можете сказать «спасибо» автору книги.

1

Введение

О книге

Если «в двух словах», то это такой CCNA по Python. С одной стороны, книга достаточно базовая, чтобы её мог одолеть любой желающий, а с другой стороны, в книге рассматриваются все основные темы, которые позволят дальше расти самостоятельно. Книга не ставит своей целью глубокое рассмотрение Python. Задача книги – объяснить понятным языком основы Python и дать понимание необходимых инструментов для его практического использования. Всё, что рассматривается в книге, ориентировано на сетевое оборудование и работу с ним. Это даёт возможность сразу использовать в работе сетевого инженера то, что было изучено на курсе. Все примеры показываются на примере оборудования Cisco, но, конечно же, они применимы и для любого другого оборудования.

Для кого эта книга

Для сетевых инженеров с опытом программирования и без. Все примеры и домашние задания будут построены с уклоном на сетевое оборудование. Эта книга будет полезна сетевым инженерам, которые хотят автоматизировать задачи, с которыми сталкиваются каждый день и хотели заняться программированием, но не знали, с какой стороны подойти.

Ещё не решили, нужно ли читать книгу? Почитайте *отзывы*.

Зачем Вам учиться программировать?

Знание программирования для сетевого инженера сравнимо со знанием английского. Если вы знаете английский хотя бы на уровне, который позволяет читать техническую документацию, вы сразу же расширяете свои возможности:

- доступно в несколько раз больше литературы, форумов и блогов;

- практически для любого вопроса или проблемы достаточно быстро находится решение, если вы ввели запрос в Google.

Знание программирования в этом очень похоже. Если вы знаете, например, Python хотя бы на базовом уровне, вы уже открываете массу новых возможностей для себя. Аналогия с английским подходит ещё и потому, что можно работать сетевым инженером и быть хорошим специалистом без знания английского. Английский просто даёт возможности, но он не является обязательным требованием.

Требуемые версии ОС и Python

Все примеры и выводы терминала в книге показываются на Debian Linux. В книге используется Python 3.7, но для большинства примеров подойдет и Python 3.x. Только в некоторых примерах требуется версия 3.6 или выше чем 3.5. Это всегда явно указано и, как правило, касается дополнительных возможностей.

Примеры

Все примеры, которые используются в книге, располагаются в репозитории. Примеры, которые рассматриваются в разделах книги, являются обучающими. Это значит, что они не обязательно показывают лучший вариант решения задачи, так как они основаны только на той информации, которая рассматривалась в предыдущих главах книги. Кроме того, довольно часто примеры, которые давались в разделах, развиваются в заданиях. То есть, в заданиях вам нужно будет сделать лучшую, более универсальную, и, в целом, более правильную версию кода. Если есть возможность, лучше набирать код, который используется в книге, самостоятельно, или, как минимум, скачать примеры и попробовать что-то в них изменить – так информация будет лучше запоминаться. Если такой возможности нет, например, когда вы читаете книгу в дороге, лучше повторить примеры самостоятельно позже. В любом случае, обязательно нужно делать задания вручную.

Задания

Все задания и вспомогательные файлы можно скачать в репозитории, том же, где располагаются примеры кода. Если в заданиях раздела есть задания с буквами (например, 5.2a), то нужно выполнить сначала задания без букв, а затем с буквами. Задания с буквами, как правило, немного сложнее заданий без букв и развивают идею в соответствующем задании без буквы. Если получается, лучше делать задания по порядку. В книге специально не приведены ответы на задания, так как, к сожалению, когда есть ответы, очень часто вместо того, чтобы попытаться решить сложное задание самостоятельно, подглядывают в них. Конечно, иногда возникает ситуация, когда никак не получается решить задание – попробуйте отложить его, задать вопрос в Slack и сделать какое-либо другое.

Примечание: На Stack Overflow есть ответы практически на любые вопросы. Так что, если Google отправил Вас на него, это, с большой вероятностью значит, что ответ найден. Запросы, конечно же, лучше писать на английском – по Python очень много материалов и, как правило, подсказку найти легко

Ответы могли бы показать, как ещё можно выполнить задание или же как лучше это сделать. Но на этот счёт не следует переживать, так как, скорее всего, в следующих разделах встретится пример, в котором будет показано, как писать такой код.

Вопросы

Для части тем книги созданы вопросы:

- Типы данных. Часть 1
- Типы данных. Часть 2
- Контроль хода программы. Часть 1
- Контроль хода программы. Часть 2
- Функции и модули. Часть 1
- Функции и модули. Часть 2
- Регулярные выражения. Часть 1
- Регулярные выражения. Часть 2
- Базы данных

Эти вопросы можно использовать как для проверки знаний, так и в роли заданий. Очень полезно поотвечать на вопросы после прочтения соответствующей темы. Они позволят вам вспомнить материал темы, а также увидеть на практике разные аспекты работы с Python. Постарайтесь сначала ответить самостоятельно, а затем подсмотреть ответы в IPython по тем вопросам, в которых вы сомневаетесь.

Презентации

Для всех тем книги есть презентации в репозитории. По ним удобно быстро просматривать информацию и повторять. Если вы знаете основы Python, то стоит их пролистать.

Скачать все презентации в формате PDF можно в специальном репозитории

Форматы файлов книги

Книгу можно скачать в двух форматах: PDF, Epub. Они автоматически обновляются, поэтому всегда содержат одинаковую информацию.

Обсуждение

Для обсуждения книги, заданий, а также связанных вопросов используется Slack. Все вопросы, предложения и замечания по книге также пишите в Slack.

Часто задаваемые вопросы (FAQ)

Здесь собраны вопросы, которые наиболее часто возникают при чтении книги.

Будет ли печатная версия книги?

Нет. Книга существует в каком-то виде с 2015 года. Все это время книга менялась. Мне нравится эта возможность менять книгу, писать что-то по-другому.

Почему в книге нет темы X?

Полезных тем еще огромное количество и просто невозможно все их вместить в одну книгу. Конечно, у каждого читателя есть приоритеты и кажется именно этот модуль очень нужен всем, но таких тем/модулей очень много. Глобально в книге уже ничего не будет менятся, новые темы добавляться не будут.

Чем это отличается от обычного вводного курса по Python?

Основных отличий три:

- основы даются достаточно коротко;
- подразумевается определённая предметная область знаний (сетевое оборудование);
- все примеры, по возможности, ориентированы на сетевое оборудование.

Я сетевик. Для чего мне нужна эта книга?

В первую очередь – для автоматизации рутинных задач. Автоматизация даёт несколько преимуществ:

- высокоуровневое мышление – проще подняться над всем, когда вы свободны от рутинной работы. У Вас появится время и возможность думать об улучшениях;
- доверие – вы не будете бояться делать изменения, которые, как правило, сопряжены с риском, так как сеть это основа работы всех приложений и цена ошибки высока;
- консистентная конфигурация – вы сможете автоматизированно создавать файлы настроек сетевого оборудования, от пользователей и подписей интерфейсов до функционала безопасности, и будете меньше переживать о том, забыли ли вы нечто.

Конечно, не будет такого, что после прочтения книги, вы «всё автоматизируете и наступит счастье», но это шаг в данном направлении. Я ни в коем случае не агитирую за то, чтобы автоматизация выполнялась кучей самописных скриптов. Если есть софт, который решает нужные Вам задачи, это отлично, используйте его. Но если его нет, или если вы просто ещё о таком не думали, попробуйте начать с простого – Ansible, например, позволит выполнять многие задачи практически «из коробки».

Зачем тогда учить Python? Дело в том, что тот же Ansible не решит все вопросы. И, возможно, вам понадобится добавить какой-то функционал самостоятельно. Кроме непосредственной настройки оборудования, есть ежедневные рутинные задачи, которые можно автоматизировать с помощью Python. Скажем так, если вы не хотите разбираться с Python, но хотите автоматизировать процесс настройки и работы с оборудованием, обратите своё внимание на Ansible. Даже «из коробки» он будет очень полезен. Если же вы потом войдете во вкус и захотите добавить своё, чего нет в Ansible, возвращайтесь :-)

И ещё, этот курс не только о том, как использовать Python для подключения к оборудованию и его настройке. Он и о том, как решать задачи, которые не касаются подключения к оборудованию, например, изменить что-то в нескольких файлах конфигурации, или обработать log-файл – Python поможет вам решать в том числе и подобные задачи.

Почему книга именно для сетевых инженеров?

Есть несколько причин:

- сетевые инженеры уже обладают опытом работы в ИТ, и часть концепций им знакома, и, скорее всего, какие-то основы программирования большинству уже будут знакомы. Это означает, что будет гораздо проще разобраться с Python;
- работа в командной строке и написание скриптов вряд ли испугает их;
- у сетевых инженеров есть знакомая им предметная область, на которую можно опираться при составлении примеров и заданий.

Если рассказывать на абстрактных примерах «о котиках и зайчиках», это одно. Но когда в примерах есть возможность использовать идеи из предметной области, всё становится проще, рождаются конкретные идеи, как улучшить какую-либо программу, скрипт. А когда человек пытается её улучшить, он начинает разбираться с новым - это очень сильно помогает продвигаться вперёд.

Почему именно Python?

Причины следующие:

- в контексте работы с сетевым оборудованием, сейчас часто используется именно Python;
- на некотором оборудовании Python встроен или есть API, который поддерживает Python;
- Python достаточно прост для изучения (конечно, это относительно, и более простым может казаться другой язык, но, скорее, это будет из-за имеющегося опыта работы с языком, а не потому, что Python сложный);
- с Python вы вряд ли быстро дойдете до границ возможностей языка;
- Python может использоваться не только для написания скриптов, но и для разработки приложений. Разумеется, это не является задачей этой книги, но, по крайней мере, вы потратите время на язык, который позволит вам легко шагнуть дальше, чем написание простых скриптов;
- из программ, связанных с сетями, на Python написан, например, GNS3.

И еще один момент – в контексте книги, Python нужно рассматривать не как единственно правильный вариант, и не как «правильный» язык. Нет, Python это просто инструмент, как отвёртка например, и мы учимся им пользоваться для конкретных задач. То есть, никакой идеологической подоплеки здесь нет, никакого «только Python» и никакого поклонения тем более. Странно поклоняться отвертке :-) Всё просто - есть хороший и удобный инструмент, который подойдет к разным задачам. Он не лучший во всём и далеко не единственный язык в принципе. Начните с него, и потом вы сможете самостоятельно выбрать нечто другое, если захотите – эти знания всё равно не пропадут.

Нужный мне модуль не поддерживает Python 3

Есть несколько вариантов решения:

- попробуйте найти альтернативный модуль, который поддерживает Python 3 (не обязательно последней версии языка);
- попробуйте найти community-версию этого модуля для Python 3. Возможно, официальной версии нет, но сообщество могло перевести его самостоятельно на версию 3, особенно если этот модуль популярен;
- используйте Python 2.7, ничего страшного не произойдет. Если вы не собираетесь писать огромное приложение, а просто используете Python для автоматизации своих задач, Python 2.7 совершенно точно подойдет.

Я не знаю, нужно ли мне это.

Я, конечно же, считаю, что нужно :-) Иначе я бы не писала эту книгу. Совсем не факт, что вам захочется погружаться во всё это, поэтому для начала попробуйте разобраться с Ansible. Возможно, вам хватит надолго его возможностей. Начните с простых команд show, попробуйте подключиться сначала к тестовому оборудованию (виртуальным машинам), затем попробуйте выполнить команду show на реальной сети, на 2-3 устройствах, потом на большем количестве. Если вам этого будет достаточно, можно остановиться на этом. Следующим шагом я бы попробовала использование Ansible для генерации шаблонов конфигурации.

Зачем сетевому инженеру программирование?

На мой взгляд, для сетевого инженера умение программировать очень важно, и не потому, что сейчас все об этом говорят, или пугают SDN, потерей работы или чем-то подобным, а потому, что сетевой инженер постоянно сталкивается с:

- рутинными задачами;
- проблемами и решениями, которые надо протестировать;
- большим объёмом однотипных и повторяющихся задач;
- большим количеством оборудования;

На текущий момент большое количество оборудования по-прежнему предлагает нам только интерфейс командной строки и неструктурированный вывод команд. Управляющий софт часто ограничен вендором, дорого стоит и имеет урезанные возможности – в итоге мы вручную снова и снова делаем одно и то же. Даже такие банальные вещи, как отправить одну и ту же команду show на 20 устройств, не всегда просто сделать. Допустим, ваш SSH-клиент поддерживает эту возможность. А если вам теперь надо проанализировать вывод? Мы ограничены теми средствами, которые нам дали, а знание программирования, даже самое базовое, позволяет нам расширить наши средства и даже создавать новые. Я не считаю, что всем надо торопиться учиться программировать, но для инженера это очень важный навык. Именно для инженера, а не для всех на свете.

Сейчас явно наблюдается тенденция, которую можно описать фразой «все учимся программировать», и это, в целом, хорошо. Но программирование это не что-то элементарное, это сложно, в это нужно вкладывать много времени, особенно если вы никогда не имели отношения к техническому миру. Может сложиться впечатление, что достаточно пройти «вот эти вот курсы» и через 3 месяца вы крутой программист с высокой зарплатой. Нет, этот книга не об этом :-) Мы не говорим в ней о программировании как профессии и не ставим такую цель, мы говорим о программировании как инструменте, таком как, например, знание CLI Linux. Дело не в том, что инженеры какие-то особенные, просто, как правило:

- они уже имеют техническое образование;
- многие работают, так или иначе, с командной строкой;

- они сталкивались, как минимум, с одним языком программирования;
- у них «инженерный склад ума».

Это не значит, что всем остальным «не дано». Просто инженерам это будет проще.

Книга будет когда-то платной?

Нет, эта книга всегда будет бесплатной. Я читаю платно онлайн курс «Python для сетевых инженеров», но это не будет влиять на эту книгу - она всегда будет бесплатной.

Благодарности

Спасибо всем, кто проявил интерес к первому анонсу курса – ваш интерес подтвердил, что это будет кому-то нужно.

Павел Пасынок, спасибо тебе за то, что согласился на курс. С вами было интересно работать, и это добавило мне мотивации завершить курс, и я особенно рада, что знания, которые вы получили на курсе, нашли практическое применение.

Алексей Кириллов, самое большое спасибо тебе :-) Я всегда могла обсудить с тобой любой вопрос по курсу. Ты помогал мне поддерживать мотивацию и не уходить в дебри. Общение с тобой вдохновляло меня продолжать, особенно в сложные моменты. Спасибо тебе за вдохновение, положительные эмоции и поддержку!

Спасибо всем, кто писал комментарии к книге – благодаря вам в книге не только стало меньше опечаток и ошибок, но и содержание книги стало лучше.

Спасибо всем слушателям онлайн-курса – благодаря вашим вопросам книга стала намного лучше.

Слава Скороход, спасибо тебе огромное, что вызвался быть редактором – количество ошибок теперь стремится к нулю :-)

2 Ресурсы для обучения по книге

Ресурсы, которые пригодятся в процессе обучения:

- репозиторий с примерами, заданиями
- репозиторий с ответами
- вопросы (мини-задания)
- в чате slack PyNEng можно задать вопросы

Подготовка рабочего окружения

Для выполнения заданий книги можно использовать несколько вариантов:

- взять подготовленную виртуалку vmware или vagrant (virtualbox)
- подготовить виртуалку самостоятельно
- использовать вм или какой-то сервис в облаке
- работать без создания виртуальной машины

Подробнее о подготовке рабочего окружения *Подготовка рабочего окружения*.

Задания

Все задания и вспомогательные файлы можно скачать в репозитории. Задания продублированы в книге исключительно для удобного обзора всех заданий раздела. Так как все вспомогательные файлы, код и тесты находятся в репозитории заданий, лучше делать задания в копии репозитория. Как сделать копию описано во втором разделе.

Иногда, какой-то раздел может оказаться особенно сложным, в этом случае, можно остановиться на минимуме заданий. Это позволит двигаться дальше и не забрасывать учебу. А позже можно вернуться и доделать задания. В целом, конечно лучше делать все задания, так как практика это единственный способ нормально разобраться с темой, но иногда лучше сделать меньше заданий и продолжать учебу, чем застрять на одной теме и забросить все.

Раздел	Минимум заданий	Все задания
04_data_structures	4.1, 4.2, 4.3, 4.6	4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8
05_basic_scripts	5.1, 5.1a, 5.2, 5.2a	5.1, 5.1a, 5.1b, 5.1c, 5.1d, 5.2, 5.2a, 5.3, 5.3a
06_control_structures	6.1, 6.2, 6.3	6.1, 6.2, 6.2a, 6.2b, 6.3
07_files	7.1, 7.2, 7.3	7.1, 7.2, 7.2a, 7.2b, 7.2c, 7.3, 7.3a, 7.3b
09_functions	9.1, 9.1a, 9.2, 9.2a, 9.3	9.1, 9.1a, 9.2, 9.2a, 9.3, 9.3a, 9.4
11_modules	11.1, 11.2	11.1, 11.2
12_useful_modules	12.1, 12.2	12.1, 12.2, 12.3
15_module_re	15.1, 15.2, 15.3, 15.4	15.1, 15.1a, 15.1b, 15.2, 15.2a, 15.3, 15.4, 15.5
17_serialization	17.1, 17.2, 17.3	17.1, 17.2, 17.3, 17.3a, 17.3b, 17.4
18_ssh_telnet	18.1, 18.1a, 18.2, 18.2a, 18.2b, 18.3	18.1, 18.1a, 18.1b, 18.2, 18.2a, 18.2b, 18.2c, 18.3
19_concurrent_connections	19.1, 19.2, 19.3	19.1, 19.2, 19.3, 19.3a, 19.4
20_jinja2	20.1, 20.2, 20.3	20.1, 20.2, 20.3, 20.4, 20.5, 20.5a
21_textfsm	21.1, 21.1a, 21.2, 21.3, 21.4	21.1, 21.1a, 21.2, 21.3, 21.4, 21.5
22_oop_basics	22.1, 22.1a, 22.1b, 22.2, 22.2a	22.1, 22.1a, 22.1b, 22.1c, 22.1d, 22.2, 22.2a, 22.2b, 22.2c
23_oop_special_methods	23.1, 23.1a, 23.2	23.1, 23.1a, 23.2, 23.3, 23.3a
24_oop_inheritance	24.1, 24.2, 24.2a	24.1, 24.1a, 24.2, 24.2a, 24.2b, 24.2c, 24.2d
25_db	25.1, 25.2, 25.3	25.1, 25.2, 25.3, 25.4, 25.5, 25.5a, 25.6

Тесты

В репозитории заданий для проверки заданий есть автоматические тесты. Они помогают проверить все ли соответствует поставленной задаче, а также дают обратный отклик по тому, что не соответствует задаче. Как правило, после первого периода адаптации к тестам, становится проще делать задания с тестами.

Как работать с тестами.

Дополнительные материалы

Почти в каждой части книги есть глава «Дополнительные материалы», в которой находятся полезные материалы и ссылки по теме, а также ссылки на документацию. Кроме того, я сделала подборку ресурсов по Python для сетевых инженеров, где можно найти много полезных статей, книг, видео курсов и подкастов.

3

I. Основы Python

Первая часть книги посвящена основам Python. В ней рассматриваются:

- типы данных Python;
- как создавать базовые скрипты;
- контроль хода программы;
- работа с файлами.

1. Подготовка к работе

Для того, чтобы начать работать с Python, надо определиться с несколькими вещами:

- какая операционная система будет использоваться
- какой редактор будет использоваться
- какая версия Python будет использоваться

В книге используется Debian Linux (в других ОС вывод может незначительно отличаться) и Python 3.7.

Ещё один важный момент – выбор редактора. В следующем разделе приведены примеры редакторов для разных ОС. Вместо редактора можно использовать IDE. IDE это хорошая вещь, но не стоит переходить на IDE из-за таких вещей как:

- подсветка кода
- подсказки синтаксиса
- автоматические отступы (важно для Python)

Всё это есть в любом хорошем редакторе, но для этого может потребоваться установить дополнительные модули. В начале работы может получиться так, что IDE будет только отвлекать вас обилием возможностей. Список IDE для Python можно можно посмотреть здесь. Например, можно выбрать PyCharm или Spyder для Windows.

Подготовка рабочего окружения

Для выполнения заданий книги можно использовать несколько вариантов:

- взять подготовленную виртуалку vmware или vagrant (virtualbox)
- подготовить виртуалку самостоятельно
- использовать вм или какой-то сервис в облаке
- работать без создания виртуальной машины

Подготовленные VM

Подготовлены виртуальные машины, в которых установлены:

- Debian 9.9
- Python 3.7 и 3.8 в виртуальном окружении
- IPython и другие модули, которые потребуются для выполнения заданий
- текстовые редакторы vim, Geany, Mu
- GNS3 для работы с сетевым оборудованием

Есть два варианта подготовленных виртуальных машин (по ссылкам инструкции для каждого варианта, в которых есть ссылки на образ и инструкция как работать с GNS3):

- vagrant
- vmware

Подготовка виртуальной машины/хоста самостоятельно

- Инструкция для подготовки Linux
- Нюансы подготовки и выполнения заданий на Windows

Список модулей, которые нужно установить:

```
pip install pytest pytest-clarity pyyaml tabulate jinja2 textfsm pexpect netmiko␣
↪graphviz
```

Также надо установить graphviz принятым способом в ОС (пример для debian):

```
apt-get install graphviz
```

Облачные сервисы

Ещё один вариант – использовать один из следующих сервисов:

- repl.it – этот сервис предоставляет онлайн-интерпретатор Python, а также графический редактор. Пример использования.
- PythonAnywhere - выделяет отдельную виртуалку, но в бесплатном варианте вы можете работать только из командной строки, то есть, нет графического текстового редактора;

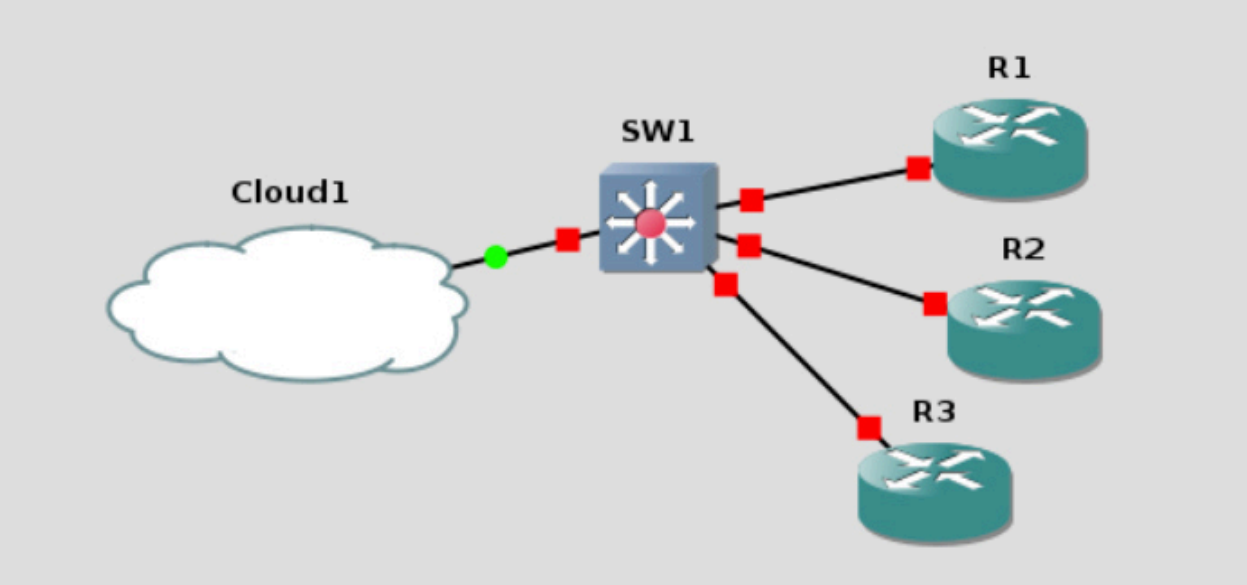
Сетевое оборудование

К 18 разделу книги, нужно подготовить виртуальное или реальное сетевое оборудование.

Все примеры и задания, в которых встречается сетевое оборудование, используют одинаковое количество устройств: три маршрутизатора с такими базовыми настройками:

- пользователь: cisco
- пароль: cisco
- пароль на режим enable: cisco
- SSH версии 2 (обязательно именно версия 2), Telnet
- IP-адреса маршрутизаторов: 192.168.100.1, 192.168.100.2, 192.168.100.3
- IP-адреса должны быть доступны из виртуалки на которой вы выполняете задания и могут быть назначены на физических/логических/loopback интерфейсах

Топология может быть произвольной. Пример топологии:



Базовый конфиг:

```
hostname R1
!
no ip domain lookup
ip domain name pyneng
!
crypto key generate rsa modulus 1024
ip ssh version 2
!
username cisco password cisco
enable secret cisco
!
line vty 0 4
 logging synchronous
 login local
 transport input telnet ssh
```

На каком-то интерфейсе надо настроить IP-адрес

```
interface ...
 ip address 192.168.100.1 255.255.255.0
```

Алиасы (по желанию)

```
!
alias configure sh do sh
alias exec ospf sh run | s ^router ospf
alias exec bri show ip int bri | exc unass
alias exec id show int desc
alias exec top sh proc cpu sorted | excl 0.00%__0.00%__0.00%
alias exec c conf t
alias exec diff sh archive config differences nvram:startup-config system:running-
↪config
alias exec desc sh int desc | ex down
alias exec bgp sh run | s ^router bgp
```

При желании можно настроить EEM applet для отображения команд, которые вводит пользователь:

```
!
event manager applet COMM_ACC
 event cli pattern ".*" sync no skip no occurs 1
 action 1 syslog msg "User $_cli_username entered $_cli_msg on device $_cli_host "
!
```

ОС и редактор

Можно выбрать любую ОС и любой редактор, но желательно использовать Python версии 3.7, так как в книге используется именно эта версия.

Все примеры в книге выполнялись на Debian, на других ОС вывод может незначительно отличаться. Для выполнения заданий из книги можно использовать Linux, macOS или Windows. Однако, стоит учитывать, что, например, Ansible можно установить только на Linux/macOS.

Для работы с Python можно выбрать любой текстовый редактор или IDE, который поддерживает Python. Как правило, для работы с Python требуется минимум настройки редактора и часто редактор по умолчанию распознает Python.

Редактор Mu

Отдельно стоит упомянуть редактор Mu: это редактор для начинающих изучать Python (он поддерживает только Python).

С одной стороны, в нём нет ничего лишнего, что поначалу может сильно отвлекать и путать. В то же время, в нём есть такие важные функции как проверка кода на соблюдение PEP 8 и debugger. Плюс, Mu работает на разных ОС (macOS, Windows, Linux).

Примечание: Записи лекций по редактору Mu: Основы работы с Mu, Использование debugger в Mu

---

IDE PyCharm

PyCharm — интегрированная среда разработки для Python. Для начинающих изучать язык может оказаться сложным вариантом из-за обилия настроек, но это зависит от личных предпочтений. В PyCharm поддерживается огромное количество возможностей, даже в бесплатной версии.

PyCharm прекрасный IDE, но, на мой взгляд, он сложноват для начинающих. Я бы не советовала использовать его, если вы с ним не знакомы и только начинаете учить Python. Вы всегда сможете перейти на него после книги, но пока что лучше попробовать что-то другое.

Geany

Geany - текстовый редактор, который поддерживает разные языки программирования, среди них Python. Также является кроссплатформенным редактором и поддерживает ОС Linux, macOS, Windows.

Примечание: Варианты редакторов выше приведены для примера, вместо них можно использовать любой текстовый редактор, который поддерживает Python.

Система управления пакетами pip

Для установки пакетов Python будет использоваться pip. Это система управления пакетами, которая используется для установки пакетов из Python Package Index (PyPI). Скорее всего, если у вас уже установлен Python, то установлен и pip.

Проверка версии pip:

```
$ pip --version
pip 19.1.1 from /home/vagrant/venv/pyneng-py3-7/lib/python3.7/site-packages/pip␣
↪(python 3.7)
```

Если команда выдала ошибку, значит, pip не установлен. Установка pip описана в документации

Установка модулей

Для установки модулей используется команда `pip install`:

```
$ pip install tabulate
```

Удаление пакета выполняется таким образом:

```
$ pip uninstall tabulate
```

Кроме того, иногда необходимо обновить пакет:

```
$ pip install --upgrade tabulate
```

pip или pip3

В зависимости от того, как установлен и настроен Python в системе, может потребоваться использовать pip3 вместо pip. Чтобы проверить, какой вариант используется, надо выполнить команду `pip --version`.

Вариант, когда pip соответствует Python 2.7:

```
$ pip --version
pip 9.0.1 from /usr/local/lib/python2.7/dist-packages (python 2.7)
```

Вариант, когда pip3 соответствует Python 3.7:

```
$ pip3 --version
pip 19.1.1 from /home/vagrant/venv/pyneng-py3-7/lib/python3.7/site-packages/pip␣
↪(python 3.7)
```

Если в системе используется pip3, то каждый раз, когда в книге устанавливается модуль Python, нужно будет заменить pip на pip3.

Также можно использовать альтернативный вариант вызова pip:

```
$ python3.7 -m pip install tabulate
```

Таким образом, всегда понятно для какой именно версии Python устанавливается пакет.

Виртуальные окружения

Виртуальные окружения:

- позволяют изолировать различные проекты друг от друга;
- пакеты, которые нужны разным проектам, находятся в разных местах – если, например, в одном проекте требуется пакет версии 1.0, а в другом проекте требуется тот же пакет, но версии 3.1, то они не будут мешать друг другу;
- пакеты, которые установлены в виртуальных окружениях, не перебивают глобальные пакеты.

Примечание: В Python есть несколько вариантов для создания виртуальных окружений. Использовать можно любой из них. Для начала можно использовать virtualenvwrapper, а затем со временем уже разбираться с тем, какие еще есть варианты.

---

virtualenvwrapper

Виртуальные окружения создаются с помощью virtualenvwrapper.

Установка virtualenvwrapper с помощью pip:

```
$ sudo pip3.7 install virtualenvwrapper
```

После установки, в файле .bashrc, находящимся в домашней папке текущего пользователя, нужно добавить несколько строк:

```
export VIRTUALENVWRAPPER_PYTHON=/usr/local/bin/python3.7
export WORKON_HOME=~/venv
. /usr/local/bin/virtualenvwrapper.sh
```

Если вы используете командный интерпретатор, отличный от bash, посмотрите, поддерживается ли он в документации virtualenvwrapper. Переменная окружения VIRTUALENVWRAPPER_PYTHON указывает на бинарный файл командной строки Python, WORKON_HOME – на расположение виртуальных окружений. Третья строка указывает, где находится скрипт, установленный с пакетом virtualenvwrapper. Для того, чтобы скрипт virtualenvwrapper.sh выполнился и можно было работать с виртуальными окружениями, надо перезапустить bash.

Перезапуск командного интерпретатора:

```
$ exec bash
```

Такой вариант может быть не всегда правильным. Подробнее на Stack Overflow.

Работа с виртуальными окружениями

Создание нового виртуального окружения, в котором Python 3.7 используется по умолчанию:

```
$ mkvirtualenv --python=/usr/local/bin/python3.7 pyneng
New python executable in PyNEng/bin/python
Installing distribute.....................done.
Installing pip..............done.
(pyneng)$
```

В скобках перед стандартным приглашением отображается имя виртуального окружения. Это означает, что вы находитесь в нём. В virtualenvwrapper по Tab работает автодополнение имени виртуального окружения. Это особенно удобно в тех случаях, когда виртуальных окружений много. Теперь в том каталоге, который был указан в переменной окружения WORKON_HOME, создан каталог pyneng:

```
(pyneng)$ ls -ls venv
total 52
....
4 -rwxr-xr-x 1 nata nata   99 Sep 30 16:41 preactivate
4 -rw-r--r-- 1 nata nata   76 Sep 30 16:41 predeactivate
4 -rwxr-xr-x 1 nata nata   91 Sep 30 16:41 premkproject
4 -rwxr-xr-x 1 nata nata  130 Sep 30 16:41 premkvirtualenv
4 -rwxr-xr-x 1 nata nata  111 Sep 30 16:41 prermvirtualenv
4 drwxr-xr-x 6 nata nata 4096 Sep 30 16:42 pyneng
```

Выход из виртуального окружения:

```
(pyneng)$ deactivate
$
```

Для перехода в созданное виртуальное окружение надо выполнить команду workon:

```
$ workon pyneng
(pyneng)$
```

Если необходимо перейти из одного виртуального окружения в другое, то необязательно делать deactivate, можно перейти сразу через workon:

```
$ workon Test
(Test)$ workon pyneng
(pyneng)$
```

Если виртуальное окружение нужно удалить, то надо использовать команду rmvirtualenv:

```
$ rmvirtualenv Test
Removing Test...
$
```

Посмотреть, какие пакеты установлены в виртуальном окружении можно командой lssitepackages:

```
(pyneng)$ lssitepackages
ANSI.py                                 pexpect-3.3-py2.7.egg-info
ANSI.pyc                                pickleshare-0.5-py2.7.egg-info
decorator-4.0.4-py2.7.egg-info          pickleshare.py
decorator.py                            pickleshare.pyc
decorator.pyc                           pip-1.1-py2.7.egg
distribute-0.6.24-py2.7.egg             pxssh.py
easy-install.pth                        pxssh.pyc
fdpexpect.py                            requests
fdpexpect.pyc                           requests-2.7.0-py2.7.egg-info
FSM.py                                  screen.py
FSM.pyc                                 screen.pyc
IPython                                 setuptools.pth
ipython-4.0.0-py2.7.egg-info            simplegeneric-0.8.1-py2.7.egg-info
ipython_genutils                        simplegeneric.py
ipython_genutils-0.1.0-py2.7.egg-info   simplegeneric.pyc
path.py                                 test_path.py
path.py-8.1.1-py2.7.egg-info            test_path.pyc
path.pyc                                traitlets
pexpect                                 traitlets-4.0.0-py2.7.egg-info
```

Встроенный модуль venv

Начиная с версии 3.5, в Python рекомендуется использовать модуль venv для создания виртуальных окружений:

```
$ python3.7 -m venv new/pyneng
```

Вместо python3.7 может использоваться python или python3, в зависимости от того, как установлен Python 3.7. Эта команда создаёт указанный каталог и все необходимые каталоги внутри него, если они не были созданы.

Команда создаёт следующую структуру каталогов:

```
$ ls -ls new/pyneng
total 16
4 drwxr-xr-x 2 vagrant vagrant 4096 Aug 21 14:50 bin
4 drwxr-xr-x 2 vagrant vagrant 4096 Aug 21 14:50 include
4 drwxr-xr-x 3 vagrant vagrant 4096 Aug 21 14:50 lib
4 -rw-r--r-- 1 vagrant vagrant   75 Aug 21 14:50 pyvenv.cfg
```

Для перехода в виртуальное окружение надо выполнить команду:

```
$ source new/pyneng/bin/activate
```

Для выхода из виртуального окружения используется команда deactivate:

```
$ deactivate
```

Подробнее о модуле venv в документации.

Установка пакетов

Например, установим в виртуальном окружении пакет simplejson.

```
(pyneng)$ pip install simplejson
...
Successfully installed simplejson
Cleaning up...
```

Если перейти в интерпретатор Python и импортировать simplejson, то он доступен и никаких ошибок нет:

```
(pyneng)$ python
>>> import simplejson
>>> simplejson
```

(continues on next page)

(продолжение с предыдущей страницы)

```
<module 'simplejson' from '/home/vagrant/venv/pyneng-py3-7/lib/python3.7/site-
↪packages/simplejson/__init__.py'>
>>>
```

Но если выйти из виртуального окружения и попытаться сделать то же самое, то такого модуля нет:

```
(pyneng)$ deactivate

$ python
>>> import simplejson
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'simplejson'
>>>
```

Интерпретатор Python

Перед началом работы надо проверить, что при вызове интерпретатора Python вывод будет таким:

```
$ python
Python 3.7.3 (default, May 13 2019, 15:44:23)
[GCC 4.9.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
```

Вывод показывает, что установлен Python 3.7. Приглашение >>>, это стандартное приглашение интерпретатора Python. Вызов интерпретатора выполняется командой python, а чтобы выйти, нужно набрать `quit()`, либо нажать Ctrl+D.

Примечание: В книге, вместо стандартного интерпретатора Python, будет использоваться ipython

---

Дополнительные материалы

Документация:

- Python Setup and Usage
- pip
- venv

- virtualenvwrapper

Редакторы и IDE:

- PythonEditors
- IntegratedDevelopmentEnvironments
- VIM and Python - a Match Made in Heaven

Задания

Задание 1.1

Единственное задание в этом разделе: подготовка к работе.

Для этого нужно:

- определиться с ОС, которую вы будете использовать
  - так как все примеры в курсе ориентированы на Linux (Debian), желательно использовать его
  - желательно использовать новую виртуальную машину, чтобы было спокойней экспериментировать
- установить Python 3.7, 3.8 или 3.9 (3.7, если будете использовать редактор Mu). Проверить, что Python и pip установлены
- создать виртуальное окружение, в котором вы будете работать весь курс
- определиться с редактором/IDE
- начиная с раздела 18, в заданиях надо будет подключаться к оборудованию. Поэтому нужно подготовить виртуальное или реальное оборудование

2. Использование Git и GitHub

В книге достаточно много заданий и нужно где-то их хранить. Один из вариантов – использование для этого Git и GitHub. Конечно, можно использовать для этого и другие средства, но используя GitHub, можно постепенно разобраться с ним и затем использовать его для других задач. Задания и примеры из книги находятся в отдельном репозитории на GitHub. Их можно скачать как zip-архив, но лучше работать с репозиторием с помощью Git, тогда можно будет посмотреть внесённые изменения и легко обновить репозиторий. Если изучать Git с нуля и, особенно, если это первая система контроля версий, с которой Вы работаете, информации может быть очень много, поэтому в этой главе всё нацелено на практическую сторону вопроса, и рассказывается:

- как начать использовать Git и GitHub;
- как выполнить базовые настройки;
- как посмотреть информацию и/или изменения.

Теории в этом подразделе будет мало, но будут даны ссылки на полезные ресурсы. Попробуйте сначала провести все базовые настройки для выполнения заданий, а потом продолжайте читать книгу. И в конце, когда базовая работа с Git и GitHub будет уже привычным делом, почитайте о них подробнее. Для чего может пригодиться Git:

- для хранения конфигураций и всех изменений в них;
- для хранения документации и всех её версий;
- для хранения схем и всех их версий;
- для хранения кода и его версий.

GitHub позволяет централизованно хранить все перечисленные выше вещи, но следует учитывать, что эти ресурсы будут доступны и другим. У GitHub есть и приватные репозитории (платные), но даже в них, пожалуй, не стоит выкладывать такую информацию, как пароли. Конечно, основное использование GitHub — размещение кода различных проектов. Кроме этого, GitHub ещё и:

- хостинг для вашего сайта (GitHub Pages);
- хостинг для онлайн-презентаций и инструмент для их создания (GitPitch);
- вместе с GitBook, это ещё и платформа для публикации книг, документации или подобного тому.

Основы Git

Git — это распределённая система контроля версий (Version Control System, VCS), которая широко используется и выпущена под лицензией GNU GPL v2. Она может:

- отслеживать изменения в файлах;

- хранить несколько версий одного файла;
- отменять внесённые изменения;
- регистрировать, кто и когда сделал изменения.

Git хранит изменения как снимок (snapshot) всего репозитория. Этот снимок выполняется после каждого коммита (commit).

Установка Git:

```
$ sudo apt-get install git
```

Первичная настройка Git

Для начала работы с Git, необходимо указать имя и e-mail пользователя, которые будут использоваться для синхронизации локального репозитория с репозиторием на GitHub:

```
$ git config --global user.name "username"
$ git config --global user.email "username.user@example.com"
```

Посмотреть настройки Git можно таким образом:

```
$ git config --list
```

Инициализация репозитория

Инициализация репозитория выполняется с помощью команды git init:

```
[~/tools/first_repo]
$ git init
Initialized empty Git repository in /home/vagrant/tools/first_repo/.git/
```

После выполнения этой команды, в текущем каталоге создаётся папка .git, в которой содержатся служебные файлы, необходимые для Git.

Отображение статуса репозитория в приглашении

Примечание: Пропускаем эту часть на Windows.

Это дополнительный функционал, который не требуется для работы с Git, но очень помогает в этом. При работе с Git очень удобно, когда можно сразу определить, находитесь вы в обычном каталоге или в репозитории Git. Кроме того, было бы хорошо понимать статус текущего

репозитория. Для этого нужно установить специальную утилиту, которая будет показывать статус репозитория. Для установки утилиты надо скопировать её репозиторий в домашний каталог пользователя, под которым вы работаете:

```
cd ~
git clone https://github.com/magicmonty/bash-git-prompt.git .bash-git-prompt --
↪depth=1
```

А затем добавить в конец файла .bashrc такие строки:

```
GIT_PROMPT_ONLY_IN_REPO=1
source ~/.bash-git-prompt/gitprompt.sh
```

Для того, чтобы изменения применились, перезапустить bash:

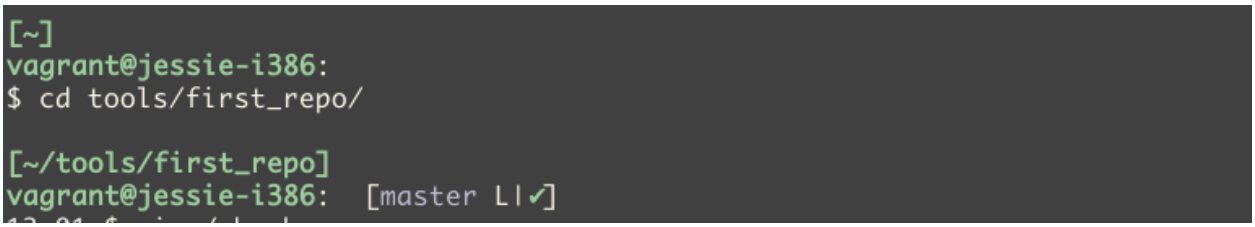
```
exec bash
```

В моей конфигурации приглашение командной строки разнесено на несколько строк, поэтому у вас оно будет отличаться. Главное, обратите внимание на то, что появляется дополнительная информация при переходе в репозиторий.

Теперь, если вы находитесь в обычном каталоге, приглашение выглядит так:

```
[~]
vagrant@jessie-i386:
$
```

Если же перейти в репозиторий Git:



Работа с Git

Для управления Git используются различные команды, смысл которых поясняется далее.

git status

При работе с Git, важно понимать текущий статус репозитория. Для этого в Git есть команда git status

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|✔]
13:02 $ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Git сообщает, что мы находимся в ветке master (эта ветка создаётся сама и используется по умолчанию), и что ему нечего добавлять в коммит. Кроме этого, Git предлагает создать или скопировать файлы и после этого воспользоваться командой git add, чтобы Git начал за ними следить.

Создание файла README и добавление в него строки «test»

```
$ vi README
$ echo "test" >> README
```

После этого приглашение выглядит таким образом

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|…2]
```

В приглашении показано, что есть два файла, за которыми Git ещё не следит

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|…2]
13:14 $ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .README.un~
        README

nothing added to commit but untracked files present (use "git add" to track)
```

Два файла получилось из-за того, что у меня настроены undo-файлы для Vim. Это специальные файлы, благодаря которым можно отменять изменения не только в текущей сессии файла, но и прошлые. Обратите внимание, что Git сообщает, что есть файлы, за которыми он не следит и подсказывает, какой командой это сделать.

Файл .gitignore

Undo-файл .README.un~ – служебный файл, который не нужно добавлять в репозиторий. В Git есть возможность указать, какие файлы или каталоги нужно игнорировать. Для этого нужно создать соответствующие шаблоны в файле .gitignore в каталоге репозитория.

Для того, чтобы Git игнорировал undo-файлы Vim, можно добавить, например, такую строку в файл .gitignore

```
*.un~
```

Это значит, что Git должен игнорировать все файлы, которые заканчиваются на «.un~».

После этого, git status показывает

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|…2]
13:33 $ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        README

nothing added to commit but untracked files present (use "git add" to track)
```

Обратите внимание, что теперь в выводе нет файла .README.un~. Как только в репозиторий был добавлен файл .gitignore, файлы, которые указаны в нём, стали игнорироваться.

git add

Для того, чтобы Git начал следить за файлами, используется команда git add.

Можно указать что надо следить за конкретным файлом

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|…2]
13:33 $ git add README
```

Или за всеми файлами

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|●1…1]
13:36 $ git add .
```

Вывод git status

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|●2]
13:36 $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   README
```

Теперь файлы находятся в секции под названием «Changes to be committed».

git commit

После того, как все нужные файлы были добавлены в staging, можно закоммитить изменения. Staging — это совокупность файлов, которые будут добавлены в следующий коммит. У команды git commit есть только один обязательный параметр – флаг «-m». Он позволяет указать сообщение для этого коммита.

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|●2]
13:37 $ git commit -m "First commit. Add .gitignore and README files"
[master (root-commit) ef84733] First commit. Add .gitignore and README files
 2 files changed, 3 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README
```

После этого git status отображает

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|✔]
13:47 $ git status
On branch master
nothing to commit, working directory clean
```

Фраза «nothing to commit, working directory clean» обозначает, что нет изменений, которые нужно добавить в Git или закоммитить.

Дополнительные возможности

git diff

Команда git diff позволяет посмотреть разницу между различными состояниями. Например, на данный момент, в репозитории внесены изменения в файл README и .gitignore.

Команда git status показывает, что оба файла изменены

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|+ 2]
13:53 $ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitignore
        modified:   README

no changes added to commit (use "git add" and/or "git commit -a")
```

Команда git diff показывает, какие изменения были внесены с момента последнего коммита

```
[~/tools/first_repo]
vagrant@jessie-i386: [master L|+ 2]
13:53 $ git diff
diff --git a/.gitignore b/.gitignore
index 8eee101..07aab05 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,2 +1,2 @@
 *.un~
-
+*.pyc
diff --git a/README b/README
index 2e7479e..79a508e 100644
--- a/README
+++ b/README
@@ -1 +1,3 @@
 First try
+
+Additional comment
```