

Оглавление

Введение	7
Вместо предисловия.....	7
Руби против ибур.....	8
Для фана.....	10
Что мы будем изучать	10
Веб-программирование или что-то другое?	11
Сколько зарабатывают программисты?	12
Ваше преимущество	13
Часть 1. Первые шаги	15
Среда исполнения	15
Настройка Windows для запуска первой программы	16
Здравствуйте, я ваш REPL	19
Запуск программы из файла.....	20
Я ваш файловый менеджер.....	21
Основы работы с файловой системой.....	24
Навигация	25
Создание файла	27
Консольный ниндзя.....	28
Текстовые редакторы	33
Первая программа.....	35
Переменные в языке Ruby	38
Сложение и умножение строк.....	40
Часть 2. Основы	42
Типы данных.....	42
Докажем, что все в Ruby – объект	44
Приведение типов (англ. converting types или type casting)	45
Дробные числа	48
Интерполяция строк.....	49
Bang!	52
Блоки	55
Блоки и параметры.....	58
Любопытные методы класса Integer	60
Сравнение переменных и ветвление	63
Комбинирование условий.....	67
Некоторые полезные функции языка Ruby	69
Генерация случайных чисел.....	71
Угадай число	74

Часть 3. Время веселья	77
Тернарный оператор	77
Индикатор загрузки	79
Методы	80
Эмулятор Судного дня.....	82
Переменные экземпляра и локальные переменные	88
Однорукий бандит (слот-машина)	90
Массивы.....	94
Немного про each	96
Инициализация массива.....	98
Обращение к массиву.....	99
Битва роботов	100
Массивы массивов (двумерные массивы)	106
Установка gem'ов	113
Обращение к массиву массивов	117
Многомерные массивы	121
Наиболее часто встречающиеся методы класса Array	122
Метод empty?	122
Методы length, size, count	124
Метод include?	125
Добавление элементов	125
Выбор элементов по критерию (select).....	126
Отсечение элементов по критерию (reject)	126
Метод take	127
Есть ли хотя бы одно совпадение (any?)	127
Все элементы должны удовлетворять критерию (all?)	127
Несколько слов о популярных методах класса Array	127
Размышления о массивах в Ruby.....	128
Символы	130
Структура данных «Хеш» (Hash)	132
Другие объекты в качестве значений.....	136
Пример JSON-структуры, описывающей приложение	138
Англо-русский словарь.....	142
Наиболее часто используемые методы класса Hash	145
Установка значения по умолчанию.....	146
Передача опций в методы.....	148
Набор ключей (HashSet)	154
Итерация по хешу	157
Метод dig	158
Проверка наличия ключа	161
Часть 4. Введение в ООП	163
Классы и объекты	163
Состояние	165
Состояние, пример программы.....	176
Полиморфизм и duck typing	181

Наследование	191
Модули.....	197
Subtyping (субтипирование) против наследования	199
Статические методы.....	204
Вся правда про ООП	208
Отладка программ	209
Отладка с использованием вывода информации в консоль	209
Отладка с использованием консольного отладчика	212
Отладка с использованием графического отладчика	218
Практическое занятие: подбор пароля и спасение мира	221
Немного про виртуализацию, Docker, основные команды Docker	234
Ruby Version Manager (RVM)	237
Тестирование	250
RSpec.....	252
Заключение.....	265
Решения задач.....	266

*Тысячи людей уже занимаются по этой книге!
Вступай и общайся в нашем ruby-сообществе в Telegram:*

<https://t.me/rubyschool>

Введение

ВМЕСТО ПРЕДИСЛОВИЯ

В XXI веке программирование стало одной из важнейших наук в любой экономике. Процессы, которые происходили раньше без помощи компьютеров, были полностью или частично оптимизированы. Бизнес и простые люди увидели пользу электронных машин, и началась эпоха расцвета IT-индустрии.

Во всем многообразии технологий образовались отдельные направления. Определились наиболее удобные инструменты для выполнения той или иной задачи. Языки программирования претерпели существенные изменения. Разобраться во всех языках и технологиях обычному читателю не так просто, как это может показаться на первый взгляд.

В какой-то момент стало очевидно, что программист – одна из профессий XXI века. Но как стать программистом? В каком направлении приложить усилия? Что нужно изучать, а что не нужно? Как наиболее эффективно использовать время, чтобы освоить какую-либо технологию?

Прежде чем дать ответ на эти вопросы, нужно ответить на самый главный вопрос: а зачем нужно становиться программистом? Какой в этом смысл?

Кто-то захочет стать программистом, чтобы разрабатывать микропрограммы для межконтинентальных баллистических ракет и космической индустрии. Кто-то хочет стать программистом для того, чтобы создавать свои собственные игры. Кто-то хочет освоить программирование в электронных таблицах, чтобы эффективнее считать налоги.

Но задача именно этой книги более бытовая. Автор подразумевает, что читатель на вопрос «зачем нужно становиться программистом?» даст ответ *«чтобы быть программистом и зарабатывать деньги»*. Обычно такой ответ дают люди, которые уже попробовали себя в какой-либо профессии и хотят более эффективно использовать свое время и получать за это деньги.

Также это могут быть молодые люди, которые вынуждены идти в ногу со временем и осваивать технологии как можно быстрее, и как можно быстрее получать результат от своих знаний. Причем результат не только в виде самих знаний – как написать ту или иную программу, а результат в денежном эквиваленте.

Знание какого-либо направления в программировании подразумевает знакомство с основами языка, с элементарной теорией (которая отличается для каждого направления), с основными понятиями и определениями, а также знакомство с неосновными инструментами (такими как операционная система, утилиты и дополнительные программы).

Направлений существует огромное множество. Это и разработка игр, и научные исследования, и обработка и анализ данных, и веб-программирование, и программирование для мобильных устройств, и т.д. Быть специалистом по всем направлениям сразу невозможно.

Поэтому человек, начинающий или желающий изучать программирование, стоит перед выбором: куда податься? что учить?

Если вы являетесь научным сотрудником НИИ, то выбор, скорее всего, падет на язык Python или C++, так как для этих языков накоплено большое количество библиотек для анализа и обработки данных.

Если вы, например, работаете сторожем и полностью довольны своей работой, то можно изучить какой-нибудь экзотический, маловостребованный на рынке язык программирования просто для того, чтобы не было скучно.

Если вы живете в обществе, где каждый месяц нужно оплачивать счета, которые становятся все больше и больше, где нужно думать не только про сегодня, но и про завтра, выбор уже будет другим. Нужно будет изучить что-нибудь быстро, очень востребованное, чтобы скорее найти работу.

Язык Руби (Ruby – англ.) и веб-программирование – это нечто среднее между *«поскорее найти работу»*, *«выучить что-нибудь несложное и интересное»* и *«чтобы также пригодилось в будущем»*. Ruby не только позволяет составлять скучные программы, работая на кого-то в офисе, но также может быть полезен дома, в быту (одна из моих последних программ – обучение игре на гитаре).

Также философия самого языка подразумевает, что обучение и использование не будет скучным. К примеру, один из принципов языка – принцип наименьшего сюрприза (principle of a least surprise), который говорит буквально следующее: *«что бы вы ни делали – скорее всего, у вас получится»*. Согласитесь, что это уже вдохновляет!

Существуют также и другие языки программирования. Автор ни в коем случае не утверждает, что они плохие. Каждый язык хорош для определенной задачи. Но вспомним про нашу задачу и сравним с некоторыми другими языками.

РУБИ ПРОТИВ ИБУР

Язык «Ибур» – это «Руби» наоборот. Это экзотический язык программирования, который, кроме меня, никто не знает. Я его сам только что придумал, и я сам не знаю, что он делает. Давайте сравним Ибур с Руби по трем параметрам, которые я описал выше.

Поскорее найти работу

Руби – очень популярный язык, легко найти работу. Ибур – никто о нем не знает, работу найти невозможно.

Остальные параметры можно не сравнивать. Другими словами, если вам важно не только программирование в себе (что тоже неплохо), но и возможность заработать в обозримом будущем, то Руби – неплохой выбор. Язык довольно популярен. Конечно, существуют и другие популярные языки программирования. Скажем, JavaScript, возможно, более популярен, но давайте сравним JavaScript и Руби.

Выучить что-нибудь несложное и интересное

Ruby – principle of a least surprise, что уже довольно неплохо. JavaScript – изначально не создавался с идеей «принципа наименьшего сюрприза». Сложнее, чем Ruby, так как является полностью асинхронным (пока поверьте мне на слово).

Докажем, что JavaScript не такой уж и простой, как может показаться на первый взгляд. Рассмотрим программу на Ruby, которая сортирует числа:

Пример: простая программа для сортировки четырех чисел в Ruby

```
[11, 3, 2, 1].sort()
```

Программа выше должна отсортировать числа 11, 3, 2, 1 в возрастающем порядке (пока не важно, если этот синтаксис вам непонятен, мы еще будем проходить эту тему). Результат работы программы на Ruby: 1, 2, 3, 11. Без сюрпризов! Но напишем ту же самую программу на JavaScript:

Пример: неправильная программа для сортировки четырех чисел в JavaScript

```
[11, 3, 2, 1].sort();
```

Синтаксис в этом случае очень похож и отличается лишь точкой с запятой (semicolon) в конце. Но каков будет результат? Не всегда JavaScript-программисты с опытом могут дать правильный ответ, ведь результат работы программы довольно неожиданный: 1, 11, 2, 3. Почему это так – это вопрос уже к истории. Но чтобы отсортировать числа в JavaScript, надо написать:

Пример: правильная программа для сортировки четырех чисел в JavaScript

```
[11, 3, 2, 1].sort((a, b) => a - b);
```

Если разобраться, то это несложно. Но вопрос в другом. Нужно ли вам на начальном этапе тратить время на такие тонкости? JavaScript вполне востребован, и каждый ruby-программист должен знать его на минимальном уровне. Но, признаться, быть full-time JavaScript-разработчиком я бы хотел только за очень большие деньги.

Может пригодиться в будущем

К тому же «чтобы также пригодилось в будущем» не очень подходит в случае с JavaScript. Язык очень динамично развивается. Знания, полученные 10 лет назад, уже не актуальны (в данном случае я говорю про популярные фреймворки – наборы инструментов). В случае с Ruby фреймворк Rails существует уже более 10 лет. Знания, полученные 10 лет назад, до сих пор применимы.

К слову, про применимость знаний стоит сделать отдельное замечание. Знания языков shell-скриптинга до сих пор применимы, через более чем 30 лет мало что изменилось. Знание основ Computer Science до сих пор применимо, на интервью и не только, эти знания практически не устаревают.

Про применимость какого-либо языка в будущем никто не может дать точных прогнозов. Однако можно посмотреть на статистику последних лет. На момент написания этой книги компания Microsoft купила за 7.5 миллиарда долларов GitHub, который был написан как раз на языке Ruby. Другими словами, язык на сегодняшний день находится в прекрасной форме. Выпускаются обновления, улучшаются скорость и синтаксис. А количество доступных библиотек позволяет быстро решить практически любую задачу (в рамках направления, которое называется веб-программирование).

Для фана

На наш взгляд, язык программирования должен не только решать какие-то бизнес-задачи, но и быть приятным в использовании настолько, чтобы им хотелось пользоваться каждый день.

К примеру, язык Java является отличным инструментом для решения бизнес-задач. Но требует к себе уважения – язык является статически типизированным (мы еще коснемся этой темы), необходимо указывать точный тип данных, с которыми производятся различные операции. Это требует времени и полностью оправдано в бизнес-среде, где лучше потратить в несколько раз больше времени на разработку, чем платить потом за ошибки.

В случае с Ruby можно написать программу быстро, «на коленке». Нет очень большой надежности (что тоже является проблемой), но многие компании, особенно стартапы, пришли к выводу, что надежность является «достаточной», а относительно невысокая скорость выполнения не является проблемой. Все это с лихвой компенсируется скоростью разработки. Ведь в современном мире часто требуется сделать что-то быстро, чтобы быстро получить инвестиции, привлечь первых пользователей, пока другие долго думают.

С личной точки зрения автора, Ruby является хорошим инструментом для того, чтобы сделать что-то свое. Какой-то свой проект, программу, которой можно поделиться с окружающими, привлечь к себе внимание или заработать денег.

Другими словами, Ruby – это эффективный, нескучный язык не только для работы, но и для себя лично – язык для романтиков.

Что мы будем изучать

Как уже было замечено ранее, существует множество направлений программирования. Каждое направление уникально и требует своих собственных навыков. На взгляд авторов, на данный момент существует два (возможно, и больше) «проверенных» направления в программировании, которые дают максимальный результат за минимальный срок. Под результатом тут понимается как денежная компенсация, так и само умение что-то сделать своими руками.

Первое направление – это мобильная разработка: программы для мобильных телефонов (Android, iPhone), планшетов (iPad) и других устройств. Второе направление – веб-программирование.

Если выбирать между мобильной разработкой и веб-программированием, то «быстрота освоения» любой из этих двух технологий по количеству вложенных усилий примерно одинакова. Однако мобильная разработка обладает своими минусами. Например, Java – язык для составления программ для Android – был уже упомянут выше. Нельзя сказать, что он является «достаточно простым» для новичка. Если честно, то с этим можно жить. В Java нет ничего такого, что является непостижимым или очень сложным.

Однако сама мобильная разработка часто подразумевает оптимизацию кода под мобильные устройства любыми средствами. Языки программирования и SDK (software development kit – набор разработчика для определенной платформы) очень часто навязывают определенный стиль разработки. И этот стиль сильно отличается от классического, объектно-ориентированного программи-

рования в сторону процедурного программирования. Процедурное программирование не всегда позволяет полностью использовать возможности языка, хотя это и не всегда важно, особенно если ваша задача – получить зарплату.

Второй момент в разработке программ для мобильных устройств заключается в том, что на данный момент существуют две основные мобильные платформы. Одна платформа принадлежит корпорации Apple, другая – Google. Как именно будут развиваться эти платформы в будущем, целиком зависит от политики этих компаний.

В случае с веб-программированием на языке Ruby все выглядит немного иначе. Сам язык разрабатывается и поддерживается сообществом программистов. Веб-фреймворк Rails, о котором мы еще поговорим, также поддерживается исключительно сообществом. Это позволяет программистам со всего света создавать удобный инструмент именно таким, каким хочется, не оглядываясь на политику какой-либо компании.

Более того, программы на языке Ruby редко исполняются на мобильных устройствах, поэтому «специально» оптимизировать их практически никогда не требуется. Ну и основное отличие Ruby от языков для мобильной разработки состоит в том, что Ruby – это динамический язык – не в том смысле, что он динамично развивается (и это тоже), а в том, что в нем присутствует так называемая *динамическая типизация данных*, о которой было уже упомянуто выше.

Основное преимущество динамической типизации по сравнению со статической – меньше правил и меньше строгости, что дает более высокую скорость разработки приложений программистом (за счет более медленного исполнения написанных программ и «достаточной» надежности). Но скорость исполнения нас не особо интересует, ведь Ruby не используется для разработки мобильных приложений, хотя может работать ключевым звеном на сервере и обеспечивать функционирование мобильных приложений для iOS, Android и т.д.).

Несомненно, существуют и другие направления в программировании, которые не были проверены авторами этой книги. Например, разработка компьютерных игр. Наверное, для того чтобы «проверить» все направления, не хватит жизни, поэтому мы оставим эту затею для пытливых умов и займемся тем, что точно востребовано на рынке, дает возможность «быстрого входа» и является более или менее интересным и нескудным.

ВЕБ-ПРОГРАММИРОВАНИЕ ИЛИ ЧТО-ТО ДРУГОЕ?

Книга «Ruby для романтиков» разделена на две части. В первой части (вы ее сейчас читаете) мы рассмотрим основы языка и его использование из т.н. командной строки. Во второй части (планируется) будет непосредственно веб-программирование и фреймворк Rails.

«Подождите, – скажет наблюдательный читатель, – ведь мы только что говорили про веб-программирование, а оно будет лишь во второй части?»

Все верно. Дело в том, что сам по себе язык Ruby является довольно мощным инструментом. Студенты ruby-школы находили работу и без знания веб-программирования. Основы языка, умение находить и использовать нужные библиотеки уже дают возможность создавать вполне полезные приложения, которые могут использоваться для обработки данных (например, веб-скрей-

пинг), для создания конфигурационных скриптов и управления операционной системой (что обязательно пригодится любому системному администратору), для работы с файлами различного формата и т.д.

Умение использовать язык для разного рода задач, не связанных с веб-программированием, дает неоспоримое преимущество перед тем, как вы начнете заниматься программированием для веба. По сути, само веб-программирование – это знакомство с определенными общепринятыми понятиями. А задачи мы будем решать уже с помощью инструмента, с которым мы научимся обращаться.

СКОЛЬКО ЗАРАБАТЫВАЮТ ПРОГРАММИСТЫ?

Этот вопрос очень важен для тех, кто в программировании совершенно не разбирается. Но прежде, чем на него ответить, я хочу сделать отступление.

Так как Ruby – это в основном язык для веб-программирования, именно ruby-программисты положили начало удаленной (*remote*, на расстоянии) работе. Культура работать над одним проектом удаленно больше всего выражена именно в веб-программировании.

Оно и понятно – для создания программного обеспечения, например, для самолетов, наверное, полезнее находиться именно в научном центре и работать рука об руку со своими коллегами из научного центра. Но в случае с веб-проектами часто не важно, где именно находится разработчик. Вклад в культуру удаленной разработки сделала и команда «37 signals», разработчики которой находятся в разных частях света и даже в разных временных зонах. Именно в «37 signals» появилась первая версия, пожалуй, самого популярного фреймворка для веб-разработки (Rails).

За последние 10 лет было доказано, что удаленная разработка возможна, что не всегда нужно держать команду программистов в одном офисе. Для любого ruby-программиста это огромный плюс. Ведь это означает, что ruby-программист не привязан к какой-то конкретной местности: можно работать на компанию в США из небольшого города, например, в Казахстане. При этом получать зарплату сильно выше любой зарплаты, которую можно получить «на месте».

Если взглянуть на статистику удаленных работ¹, то язык Ruby находится вверху списка по количеству доступных вакансий. Первое место, похоже, удерживает JavaScript, но только лишь из-за того, что минимальные знания JavaScript являются необходимостью и он требуется в совокупности с остальными языками: Java, PHP, Ruby и т.д. А вот «чистый JavaScript» для full-stack программирования (Node.js) хоть и востребован, но не находится в самом верху списка.

Хочется заметить, что количество работ по определенному языку не является самым важным показателем, и вообще не может быть никаких важных показателей, с помощью которых можно сделать «точный выбор» на всю оставшуюся жизнь. Мы лишь говорим о том, что мы знаем сейчас. Прогнозировать на несколько лет вперед в IT-индустрии очень сложно. Но, несомненно, хорошая новость заключается в том, что вам не нужны тысячи работ – достаточно

¹ <https://remoteok.io/stats.php>.

найти одну. Также обычно не очень важно, сколько именно времени вы потратите на поиск работы – одну неделю, две недели или два месяца.

Тут мы подходим к статистике, которая была собрана студентами ruby-школы. Так сколько же зарабатывают ruby-программисты? Прежде чем ответить, сделаем оговорку, что речь будет идти только про удаленную работу. Рынок удаленных зарплат более стабилен, он был уравновешен программистами из разных стран, и на нем сформировалась определенная цена. Нет смысла сравнивать зарплату «на месте», так как ruby-программист может (и даже обязан) работать удаленно, и в большинстве случаев это более выгодно. Также подразумевается, что программист имеет минимальные знания английского языка, которые позволяют ему общаться по переписке с заказчиками из других стран.

Категории зарплат можно условно разделить на три части. В настоящее время стоимость часа работы программиста с 1 годом опыта составляет не более 10 долларов в час. От 1 года до 3 лет – примерно от 10 до 25 долларов в час. От 3 до 7 лет – примерно от 25 до 40 долларов в час. При достижении цифры в 40 долларов в час все становится очень индивидуально. К слову, стандартное количество часов в месяц – 160.

Из нашего опыта, вполне реально без особых навыков за 1 год освоить программирование на Ruby и найти первую удаленную работу. Возможно, потребуется предрасположенность (этот факт не был доказан) и знание или желание выучить английский. Этот путь прошли многие студенты ruby-школы, и подтверждение этим словам можно найти в нашем чате¹.

ВАШЕ ПРЕИМУЩЕСТВО

Прежде чем мы приступим к созданию вашей первой программы, важно будет упомянуть о том, что к программированию не относится. Любой человек имеет разный набор жизненного опыта. Возможно, кто-то пришел в программирование из музыки, кто-то – из финансов. Любому музыканту будет в разы проще написать программу для обучения людей нотной грамоте. Финансисту будет проще написать программу для учета торгового баланса. В чем заключается ваше преимущество?

По мере изучения языка Ruby постоянно будет возникать вопрос о создании вашей собственной программы или серии программ по вашим идеям. Это необходимо по следующим причинам.

Во-первых, любая программа обычно решает какую-то бизнес-задачу. Программистам платят деньги за то, что они оптимизируют бизнес-процессы, упрощают реальную жизнь, сокращают время, которое люди тратят на какие-либо действия. Например, представьте себе очередь в каком-нибудь государственном учреждении в 1986 году. Много людей собрались в зале ожидания и ждут своей очереди. А теперь представим, что есть программист, который написал программу «электронная очередь». Через сеть Интернет любой человек может записаться на прием, прийти ровно к назначенному времени, а время, которое он провел в очереди, он потратит, например, преподавая урок математики школьникам.

¹ <https://t.me/rubyschool>.

Экономическая выгода очевидна: время, проведенное в очереди, теперь тратится с пользой. А все из-за того, что был создан какой-то полезный сайт. То же самое и с вашими знаниями. Знания какой-либо предметной области уже являются ценным активом. Попробуйте увидеть ваше преимущество, подумайте о том, каким образом вы могли бы улучшить мир. Хорошо, если у вас будет несколько идей, записанных на бумаге. По мере работы с этой книгой вы можете к ним возвращаться и задавать себе вопрос: а могу ли я это реализовать с помощью Ruby?

Во-вторых, используя свое преимущество в какой-либо области, вы сможете создавать программы просто для демонстрации своих знаний. Даже самая простейшая программа, которую может написать профессиональный музыкант, будет вызывать восторг у программистов с большим опытом, которые музыкантами не являются.

Не выбрасывайте свои программы, даже самые наивные из них можно будет в будущем улучшить. Они также пригодятся, когда вы будете искать работу; иметь на руках хоть какой-то образец кода намного лучше, чем вообще его не иметь. Ваши программы могут казаться незначительными, но при приеме на работу играет роль не отдельная программа, а совокупность всего, что вами было продемонстрировано: знания программирования, написанные программы, резюме, знания предметной области, активный GitHub-аккаунт, активный блог по программированию в Интернете.

В-третьих, если вы не работаете над своим проектом, то ваш успех зависит от случайности. Сложно предсказать, в какой именно коллектив вы попадете, какие стандарты качества создания программных продуктов будут в вашей компании. Человеку свойственно надеяться на лучшее, но практика показывает, что в реальной жизни все немного иначе и успех часто зависит от случайности.

Досадно попасть в компанию с бюрократическими сложностями, досадно попасть в коллектив с низкой технической квалификацией. Более того, начинающий программист может даже не распознать эти признаки, и как следствие – депрессия и разочарование в выбранном пути. Но на самом деле программирование должно доставлять удовольствие. И свой собственный проект – это ваш ориентир, показатель роста вашего уровня и страховка от случайности.

В любой сложной ситуации на вашей новой работе вы сможете сказать себе *«да, может быть, я не очень продуктивен на этой работе, но вот мой проект и вот демонстрация моей технической квалификации. Скорее всего, дело не во мне, а в чем-то другом»*. Более того, этот аргумент можно всегда использовать для диалога с вашим менеджером, а сам проект добавить в резюме. Ваш проект зависит только от вас, и существует отличная от нуля вероятность, что ваш собственный проект начнет приносить вам деньги.



Задание

Заведите блокнот с идеями. Записывайте туда абсолютно все идеи, которые приходят вам в голову. Возможно, вы вернетесь к ним через неделю, месяц или год.

Часть 1

Первые шаги

СРЕДА ИСПОЛНЕНИЯ

Среда исполнения – важное понятие. В дальнейшем вводится понятие *среда/окружение* (environment), но это не одно и то же. Среда исполнения – это где и «кем» будут запускаться ваши программы на языке Ruby. Скажем, ученый-химик может делать эксперимент в пробирке, в большой стеклянной банке и даже в собственной ванной. То же самое справедливо и для программы на Ruby. Она может быть исполнена разным «интерпретатором» (программой для запуска программ), в разных условиях – на операционной системе Windows, Mac, Linux.

Когда автор этих строк впервые познакомился с компьютером, среда исполнения была одна – не было никакого выбора. При включении компьютера был виден курсор и надпись «ОК», которая означала, что можно вводить программу. Сейчас компьютеры стали более умными, и новичку еще предстоит разобраться, как запускать программу, где вводить текст программы, «чем» запускать написанную программу, какая среда исполнения лучше.

Кстати, в какой именно операционной системе запускается программа, для нас не очень важно. На сегодняшний день программу, написанную на любом из популярных языков программирования, можно запустить на трех ОС: Windows, MacOS, Linux. Обычно не требуется никаких изменений в самой программе или эти изменения минимальны.

Статистика использования операционных систем показывает, что наиболее популярной ОС на сегодняшний день является ОС Windows. Именно с Windows мы и начнем, хотя это и не является лучшим выбором. Причина нашего решения в том, чтобы максимально быстро ввести вас в курс дела и любой начинающий программист максимально быстро смог написать нужную программу. Ведь настройка среды исполнения – обычно не очень простое дело для начинающих, и кажущаяся «сложность» может отпугнуть студента на первом этапе.

Несмотря на то что мы начнем запускать наши программы в ОС Windows, в будущем настоятельно рекомендуется не использовать ОС Windows для запуска программ на языке Ruby. Однако эту ОС при желании можно использовать для написания программ. В любом случае, авторы рекомендуют как можно быстрее установить Linux (Mint Cinnamon edition как наиболее простой дистрибутив) и использовать его. Если вы используете Mac, то нет необходимости устанавливать Linux.

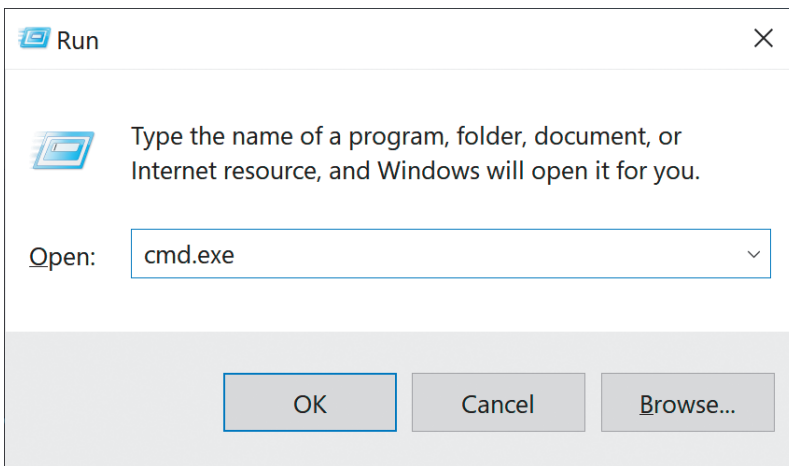
НАСТРОЙКА WINDOWS ДЛЯ ЗАПУСКА ПЕРВОЙ ПРОГРАММЫ

Терминал (который также называют словами «консоль», «оболочка», «шелл», «командная строка») – друг любого ruby-хакера. Чтобы запускать программы, которые мы с вами напишем, нужен какой-то центральный пульт, *откуда* мы будем руководить процессом. Этим пультом и служит терминал.

Ради точности следует заметить, что *терминал* – не совсем правильное слово. Но оно часто используется. Программисты говорят «запустить в терминале», но если копнуть глубже, то терминал – особая программа, которая запускает оболочку (shell). И на самом деле мы отправляем команды в оболочку, где терминал служит лишь транзитным звеном, удобной программой для соединения с оболочкой.

Забегая вперед, хочется заметить, что существуют разные типы оболочек. Стандартной оболочкой в индустрии является bash. Однако авторы рекомендуют использовать zsh (читается как «зи-шелл»), в вариации «*Oh My Zsh*»¹. Эта оболочка немного отличается от стандарта, но дает более широкие возможности и является более удобной.

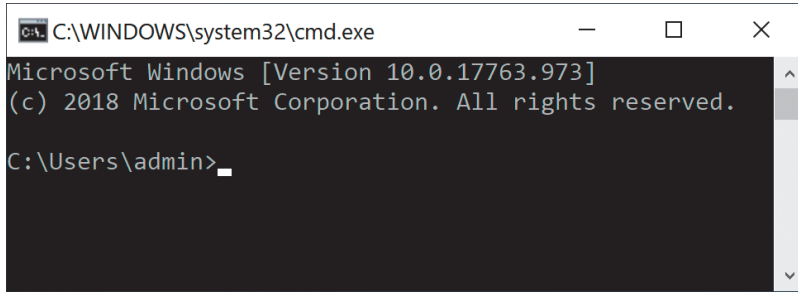
Однако в ОС Windows стандартная оболочка – это cmd.exe. Если вы нажмете Пуск – Выполнить – cmd.exe:



Запуск cmd.exe на Windows

– вы увидите черный экран и «приглашение» командной строки:

¹ <https://ohmyzsh/>.



Windows shell

«Приглашение» заканчивается символом `>`, который означает, что оболочка ожидает вашего ввода. Стоит сразу запомнить неочевидный момент: если что-то не получается, необходимо попробовать перезапустить оболочку. Это справедливо и для других операционных систем, и за свою карьеру авторы наблюдали «магическое действие» этого трюка на уже, казалось бы, очень опытных программистах. Выйти из оболочки можно словом `exit` или просто нажав на крестик вверх окна.

В ОС Linux и Mac терминал обычно доступен по умолчанию среди программ и можно запустить его, щелкнув по невзрачной иконке, скорее всего в виде прямоугольника. В этих операционных системах приглашение командной строки принято обозначать символом доллара `$`. Это не всегда правда, но на будущее стоит запомнить: если вы видите знак доллара где-нибудь в документации и после этого знака идет команда

```
$ ls
```

– то знак доллара обычно вводить не надо. Это просто индикатор того, что команду надо выполнять в оболочке `bash` (или частично совместимой с ней `zsh`).

Не важно, в какой оболочке вы сейчас находитесь, введите команду `ruby` и нажмите `Enter`. В случае с Linux и MacOS ошибки не будет, команда запустится и тихо будет ожидать окончания ввода программы. В Windows должна быть ошибка, ведь язык Ruby по умолчанию не установлен, а это значит, что нам надо его установить.

Тут следует сделать отступление. Сейчас и в будущем: если вы не знаете, что делать, задайте вопрос google. Например, в нашем случае – «*how to run ruby program on windows*». Умение задавать вопрос и искать ответ – половина дела. Если честно, то только благодаря этому умению можно научиться программировать. Главное – мыслить последовательно и логически. Если не получается, всегда можно обратиться за помощью в [чат](#)¹.

Для запуска программ на Ruby из ОС Windows нужно запустить [Ruby Installer](#)². После того как программа установлена, можно вводить команду `ruby` в терминале. Если команда не работает, попробуйте перезапустить терминал. Ruby запустится «тихо» и будет ожидать вашего ввода. Введите `puts 1+1`, затем нажмите `Enter`, а потом `Ctrl+D` (иногда `Ctrl+D` приходится нажимать два раза):

¹ <https://t.me/rubyschool>.

² <https://rubyinstaller.org/>.


```
$ ruby
puts 1+1 (нажмите Ctrl+D в этом месте)
2
$
```

Что мы видим на экране выше? Приглашение командной строки \$, вводим ruby, потом puts 1+1, потом Enter, который переводит нас на следующую строку, на которой мы нажимаем Ctrl+D. После этого «сама появляется» цифра 2. Что же тут произошло?

Во-первых, вы запустили программу для запуска программ. Ruby – это программа (интерпретатор), которая позволяет запускать ваши, человеком написанные программы. Компьютер говорит на языке нулей и единиц, и чтобы вас понять, ему надо считать человеческий язык – puts 1+1.

Комбинация Ctrl+D (обозначается также ^D) пригодится вам во всей вашей дальнейшей жизни, она передает сигнал о том, что «ввод закончен» (конец ввода, end of input, end of file, EOF). Это байт (его значение равно 4 – это запоминать не надо), который говорит о том, что наступил конец текстового потока данных, данных больше не будет. Интерпретатору ruby ничего больше не остается – только запустить то, что вы написали, что и было сделано.

Набранная вами команда puts 1+1 – это ваша первая программа. Но мы не сохраняли ее в файле, мы ввели эту программу с клавиатуры, и она «пропала» после того, как была выполнена. Сожалеем, что вы не сохранили свою первую программу. Но ничего страшного, она занимала всего лишь 8 байт, и восстановить ее – небольшая проблема. Так что же такое puts 1+1?



Задание

Прежде чем ответить на этот вопрос, выполните задание.

Запустите программу (без puts)

```
1+1
```

Мы увидим, что ничего не происходит. На самом деле результат был посчитан, но просто не выведен на экран. Возможен вариант, когда вы зададите компьютеру какую-нибудь сложную задачу и он будет считать ее очень долго. Но если вы не написали puts, то результат мы не узнаем.

Другими словами, puts выводит результат. Это сокращение от двух английских слов: *put string* (вывести строку). В других языках были приняты другие сокращения для вывода строки, например в языке BASIC это print.

Так почему же надо писать puts в начале, а не в конце? Ведь сначала надо посчитать, а потом уже выводить. Все просто, в этом случае говорят: «метод (функция) принимает параметр». То есть сначала мы говорим, что мы будем делать – *выводить*, а потом – что именно мы хотим выводить. Нашу программу можно также записать как puts(1+1). В этом случае видно, что в скобках – параметр. Ведь в математике мы сначала считаем то, что в скобках, а потом уже выполняем остальные действия. Кстати, наши поздравления! Вы написали свою первую программу.



Задание

Остановитесь тут и попробуйте написать программу, которая считает количество миллисекунд в сутках.

Следующий абзац содержит ответ:

```
$ ruby
puts 60 * 60 * 24 * 1000
(нажмите Ctrl + D)
```

Задача чисто математическая, количество секунд в минуте умножаем на количество минут в часе, умножаем на количество часов в сутках. И чтобы получились миллисекунды, а не секунды, умножаем на 1000. Далее попробуйте запустить следующую программу:

Программа Ruby для вычисления математического выражения, упомянутого выше

```
puts 5**5 * 4**4 * 3**3 * 2**2 * 1**1
```

Запись `**` означает возведение в степень. Например, $3^{**} 2 = 3 * 3 = 9$. Удивительно, но результат работы программы (5 в пятой степени, умноженное на 4 в четвертой, и т.д.) выше будет равен количеству миллисекунд в сутках! Объяснений этому нет, просто забавный факт. В качестве упражнения попробуйте запустить следующую программу:

Попробуйте угадать, что будет напечатано на экране?

```
puts 60 * 60 * 24 * 1000 == 5**5 * 4**4 * 3**3 * 2**2 * 1**1
```

ЗДРАВСТВУЙТЕ, Я ВАШ REPL

В случае с `1+1` выше наш интерпретатор выполняет два действия: `read` (прочитать), `evaluate` (выполнить). Так как не было третьего действия `print` (`puts` в нашем случае), то не было и результата на экране. То есть чтобы мы видели результат, надо выполнить:

- `read` (R);
- `evaluate` (E);
- `print` (P).

Хорошо бы еще и не запускать `ruby` каждый раз, чтобы программа в бесконечном цикле (`loop - L`) спрашивала нас «*что хотите выполнить?*», т.е. сразу принимала бы ввод без лишних разговоров.

Из начальных букв у нас получилось REPL – `read evaluate print loop`. То есть REPL – это такая программа, которая сначала читает, потом исполняет, потом печатает результат и затем начинает все сначала. Это понятие широко известно и используется не только в Ruby. А в Ruby REPL-программа называется `irb` (`interactive ruby`).

Попробуйте ввести `irb` и посмотрите, что произойдет:

```
$ irb
2.5.1 :001 >
```

Непонятные цифры в начале – это версия Ruby. В нашем случае 2.5.1 (то же самое покажет команда `ruby -v`). 001 – это номер строки. То есть если REPL уже содержит «P» (`print`), то можно вводить `1+1` без `puts`.



Задание

Посчитайте количество секунд в сутках, не выводя результат на экран с помощью `puts`.

Принцип наименьшего сюрприза говорит нам о том, что выход из REPL должен быть командой `exit`. Вводим `exit` – получилось!

Тут хочется заметить, что авторы редко используют именно `irb` в роли REPL. Есть лучшая альтернатива под названием `Pry`¹. Он выполняет ту же самую функцию, но имеет больше настроек. Этот инструмент рассматривается дальше в нашей книге.

ЗАПУСК ПРОГРАММЫ ИЗ ФАЙЛА

Запуск программы из файла ненамного сложнее. Достаточно передать аргумент интерпретатору Ruby с именем файла:

```
$ ruby app.rb
```

В этом случае интерпретатор считает программу из файла `app.rb` и запустит ее так же, как если бы вы ввели эту программу и нажали `Ctrl+D`.

Но возникают вопросы: как и где сохранить эту программу, в чем ее набрать, какой редактор кода использовать? Для начала ответим на первый вопрос – «где» сохранить программу, так как этот вопрос подразумевает знакомство с файловой системой и в нем есть некоторые подводные камни.

Для Windows, операционной системы, с которой вам нужно как можно скорее уходить на Linux, необходимо создать директорию (каталог, папку) в разделе C: и назвать ее, например, *projects*. После этого нужно перейти в директорию, создать там файл и запустить его.

Другими словами, нужно уже уметь делать как минимум четыре вещи:

- 1) создавать директорию;
- 2) переходить в директорию;
- 3) создавать файл в директории и сохранять что-то в этот файл;
- 4) запускать файл (это мы уже умеем: `ruby app.rb`).

Тут можно было бы дать основные команды ОС Linux для этих целей и не связываться на ОС Windows. Однако рынок диктует свои условия – большинство пользователей сейчас работают на Windows, а значит, с большой долей вероятности и у вас установлена эта операционная система. Но не стоит отчаиваться, мы исправим этот досадный факт, а пока постараемся как можно быстрее настроить нашу среду исполнения, чтобы мы могли писать и запускать программы, а исправлением займемся потом.

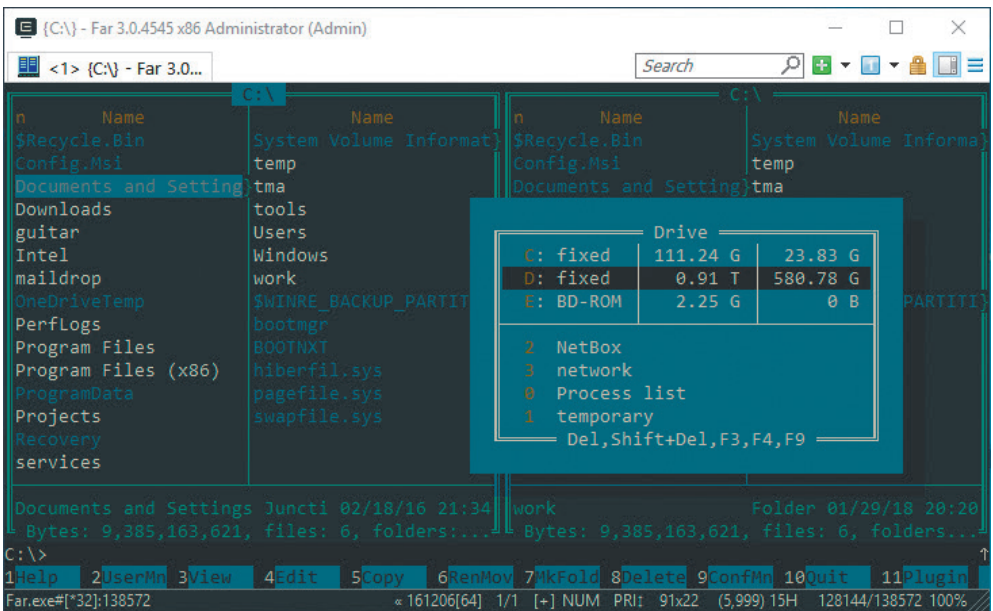
¹ <http://pry.github.io/>.

Умение ориентироваться в файловой системе – ключевой навык любого программиста. Как библиотекарь должен знать, где какая книга лежит, так и программист должен знать (или уметь разобраться, найти), где лежит тот или иной файл. Нужно всегда иметь в голове примерную «картинку» файловой системы.

Но из практики обучения студентов этому, казалось бы, простому делу выяснилось, что не все представляют себе, что такое файловая система и как эффективно работать с файлами (создавать, находить, переносить, переименовывать). Можно было бы написать список команд и дать задание запомнить эти команды. Но мы пойдем более гуманным и проверенным путем – мы познакомимся с файловым менеджером.

Я ВАШ ФАЙЛОВЫЙ МЕНЕДЖЕР

Если вы занимаетесь программированием более 20 лет, то вряд ли существует много инструментов, которые были актуальны тогда и сейчас. Именно поэтому мы изучаем Ruby, т.к. знаем, что знания, полученные 10 лет назад, до сих пор не теряют свою ценность. Но существуют также и другие инструменты, которые используются и сейчас, и пережили при этом не одно поколение операционных систем. Один из таких инструментов – файловый менеджер:

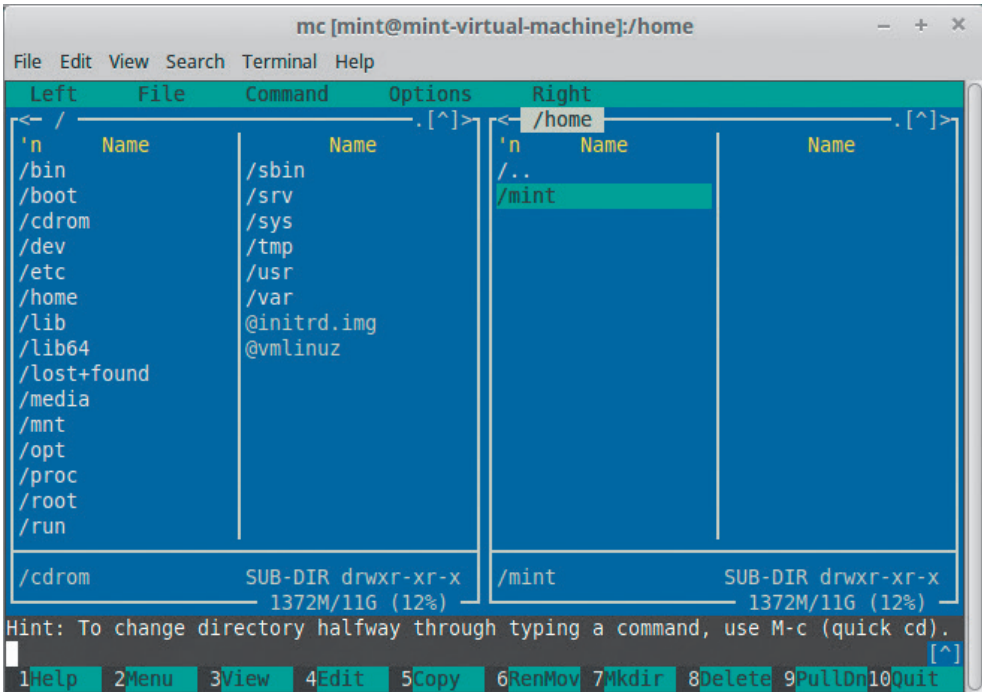


Far Manager запущен на Windows

Работа с файлами – ключевой навык программиста, сисадмина или даже любого энтузиаста. В большинстве книг по программированию работа с файлами не освещается достаточно хорошо. Дается набор шелл-команд, но никто не говорит, как работать с файлами эффективно, быстро и просто. Последнее немаловажно для любого начинающего, ведь наша задача – как можно эффективнее потратить наше время на наиболее значимые вопросы программирования, получить ра-

боту, а потом уже «дотачивать» навык. Поэтому запоминать команды оболочки, приведенные ниже, не стоит, они запомнятся сами. Более того, команды будут даны для ОС Linux (точнее, для оболочки, совместимой со стандартной bash). А комбинации клавиш – для ОС Windows, т.к. Far работает только в Windows.

«Подождите, – скажет внимательный читатель, – мы хотим уйти от Windows, но и хотим научиться работать в Far?» Дело в том, что файловый менеджер – вещь универсальная. Еще во времена DOS (уже малоизвестная операционная система от Microsoft) появился один из самых первых файловых менеджеров – *Norton Commander*. Под операционной системой Linux (а также и MacOS) существует *Midnight Commander*:



Midnight Commander запущен на Linux

Да и кроме «синих экранов» существуют различные варианты файловых менеджеров на любой вкус и цвет. Однако популярность Far'a настолько высока (из-за удобства прежде всего), что некоторые программисты нашли способ запустить его на Linux и Mac без использования эмулятора. Способ установки Far на Linux и MacOS описан по [ссылке](#)¹. Начинающие программисты могут столкнуться с трудностями, следуя инструкциям по этой ссылке, но если у вас есть опыт или время, мы настоятельно рекомендуем установить Far на Linux/MacOS. На MacOS этот файловый менеджер устанавливается одной командой:

```
$ brew install yurikoles/yurikoles/far2l
```

¹ <https://github.com/elfmz/far2l>.

Если на вашей MacOS не установлен [HomeBrew](https://brew.sh/)¹, то потребуется установить сначала эту программу. После установки вы сможете запустить файловый менеджер командой `far2l`.

Far Manager запущен на MacOS



Задание

Если вы используете MacOS или Linux, найдите и установите файловый менеджер. Пример запроса в google: «file manager for mac os». Для Linux-семейства Ubuntu установка обычно сводится к двум командам в терминале:

```
$ sudo apt-get update
$ sudo apt-get install mc
```

После этого можно вводить `mc`, чтобы запустился Midnight Commander.

Не отчаивайтесь, если у вас ничего не получилось. Имейте в виду, что если что-то не работает локально, можно всегда воспользоваться облаком. Например, сайт [Repl.it](https://repl.it/)² предлагает на выбор множество языков программирования, которые можно запустить прямо в вашем браузере. Среди этих языков есть и Ruby. Конечно, это не путь настоящего джедая, но как бэкап-план – отличное решение!

¹ <https://brew.sh/>.

² <https://repl.it/>.

ОСНОВЫ РАБОТЫ С ФАЙЛОВОЙ СИСТЕМОЙ

Говорят, что файловая система «древовидная», то есть ее можно представить в виде дерева. Каждая ветвь – это директория, в которой может быть одна или несколько других директорий (ветвей) или файлов (листьев). Также директория может быть пустой. Самую главную директорию называют «корневой» (root directory – не надо путать с root home directory – это директория пользователя с именем root).

Уже тут начинаются разногласия. Почему структура древовидная, а главная директория корневая, а не стволовая? Ведь ветви растут от главного ствола!

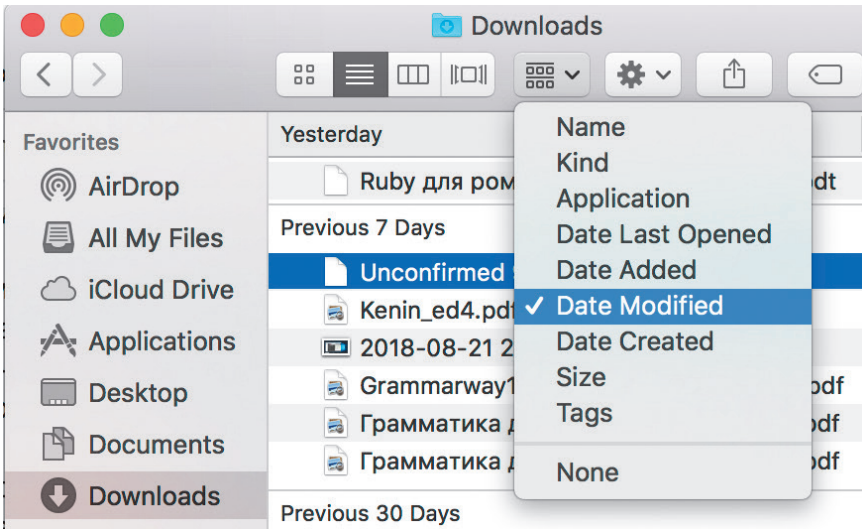
Также когда мы представляем дерево, мы подразумеваем, что дерево растет вверх. Хотя во всех файловых менеджерах «корни» растут вниз – надо нажать кнопку вниз, чтобы поставить курсор на одну директорию ниже. Может, тогда лучше говорить, что структура не древовидная, а корневидная?



Детское творчество в одном из детских садов в Кремниевой Долине. Любопытный программист задал вопрос: а где корень у этого дерева? Дело в том, что корневой, самый главный, узел (обычно обозначается как root – корень) находится в самом верху. Или мы все-таки говорим про ветви, которые растут снизу вверх? В этом вопросе есть неопределенность, пусть она вас не пугает

В любом случае, корневидная она или древовидная, у дерева вверху намного меньше ветвей, а внизу намного больше. В файловой системе такого нет, все директории и файлы по умолчанию отсортированы в алфавитном порядке. Наверное, у человека не нашлось более точной аналогии, и было принято называть файловую структуру «древовидной».

Кстати, одна из моих любимых сортировок файлов и директорий в любом файловом менеджере – по дате обновления в убывающем порядке. Такой порядок позволяет в самом верху видеть файлы, которые были обновлены недавно. А человек обычно всегда работает с самыми «свежими» файлами. Как только вы прочувствуете это преимущество, вам не захочется сортировать по-другому.



Сортировка по дате обновления в убывающем порядке в Finder (MacOS)

Те программисты, которые постоянно используют консоль (не пользуются файловыми менеджерами), упускают это очевидное преимущество. Нам достаточно скачать проект любой сложности, и мы уже будем видеть, какие директории и файлы были изменены недавно – над чем идет работа, в курсе чего нужно быть. Так что, пока не узнав ни одной команды, вы уже получили знание, которое будет полезно и которое также приобретается и осознается не сразу даже опытными программистами – озарение может наступить через несколько лет.

НАВИГАЦИЯ

Навигация в файловой системе – это просто переход из одного каталога в другой, чтобы посмотреть, что там находится, какие файлы. В Far для навигации используются кнопки вверх, вниз, Enter (войти в директорию), Tab (для перехода на соседнюю панель).

В bash для навигации существуют следующие команды (cd работает в Windows, а вот ls уже не работает):

```
$ cd my_dir # войти в директорию my_dir
$ cd .. # подняться на уровень выше
$ ls # листинг (список файлов)
```

С cd вроде бы все понятно, но листинг обычно не выдает полный список файлов. Оказывается, что есть еще скрытые файлы (в Linux и MacOS они начинаются с точки)! Поэтому команду нужно изменить на ls -a, чтобы вывести все файлы. В Far'e тоже есть такая настройка (в верхнем меню Options-Panel Settings-Show hidden and system files).

Авторы редко используют ls или ls -a. Наиболее удобная команда консоли для вывода всех файлов – это ls -lah:

- флаг `l` указывает на то, что нам нужен вывод в виде расширенного списка (который содержит права на доступ к файлу, имя владельца, размер в байтах, дату обновления);
- флаг `a` говорит о том, что надо выводить информацию обо всех файлах (all), в т.ч. скрытых;
- флаг `h` говорит о том, что нужно выводить размер файла не в байтах, а в human-readable формате, т.е. в формате, который понятен человеку (килобайты, мегабайты, гигабайты и т.д.).

Кстати, флаг `h` очень полезный и часто используется для других команд. Например, `df -h` (disk filesystem in human-readable format) выводит статистику свободного места на разделах вашего диска в гигабайтах.

```

Last login: Tue Dec 18 18:08:37 on ttys002
ro@loft ~ df -h
Filesystem      Size  Used Avail Capacity iused   ifree %used  Mounted on
/dev/disk1     465Gi 199Gi 266Gi    43% 2824643 4292142636    0% /
devfs          182Ki 182Ki   0Bi   100%    628         0 100% /dev
map -hosts      0Bi   0Bi   0Bi   100%     0         0 100% /net
map auto_home   0Bi   0Bi   0Bi   100%     0         0 100% /home
  
```

Системная информация в терминале

А в Far'e нужно для этого нажать `Ctrl+L`. Чтобы скрыть, нужно еще раз нажать `Ctrl+L`.

```

/Users/ro/work/rubyisforfun/manuscript/images} - Far 2.1 -alpha x64
  Name      Name
  ..
  009-cmd-exe.png
  009-run.png
  012-far.png
  012-far-mac.png
  012-mc.png
  013-sort-order.png
  013-tree.png

Information 20:24
Computer name      loft
User name         ro

Total bytes       499 055 067 136
Free bytes        284 272 103 424
Serial number     0100-0004

Memory
Memory load      99%
Total memory     15 455 535 104
Free memory      276 107 264

On branch master
Your branch is up-to-date with 'origin'
nothing to commit, working tree clean

Folder description file is absent

ng ro      staff      7745 15/12/18 17:29
1 267 271 bytes in 7 files
  
```

Правая панель с системной информацией в Far Manager

Вообще, программисты – довольно ленивые люди, поэтому чтобы что-то включить, а потом выключить (toggle), иногда нужно нажать одну и ту же комбинацию клавиш. Например, просмотр файла в Far и Midnight commander – это клавиша F3, а выход из просмотра – тоже F3 (кто не знает – тот обычно использует Escape и тянется в конец клавиатуры).

Мы изучили пару шелл-команд (на самом деле их не надо запоминать, просто сделайте пометку в книге), но в файл-менеджере это всего лишь несколько правильных кнопок, которые позволяют не только легче понять, как выглядит ваш проект или файловая система целиком, но и дают более наглядный результат.

Авторы книги, несмотря на большой опыт программирования и привычку делать все из «черного экрана» – консоли, время от времени все-таки запускают файловый менеджер. Особенно это полезно делать на новом проекте, который содержит много файлов.



Задание

«Походите» по вашей файловой системе и посмотрите, какие файлы и директории в ней существуют. Директории из корневого каталога («директории из корневой директории») будут часто встречаться в будущем.

Некоторые полезные горячие клавиши файлового менеджера (см. также нижнюю панель на скриншоте выше):

- F3 – переключить режим просмотра файла;
- F4 – редактировать файл;
- F5 – скопировать файл или директорию из текущей панели в другую;
- F6 – переместить файл или каталог с текущей панели на другую;
- F7 – создать каталог;
- F8 – удалить файл или каталог;
- F9 – опции;
- Tab – переход с одной панели на другую;
- Ctrl+L – переключить информационную панель системы.

Как видите, основные операции с файлами выполняются с помощью F-клавиш. Если вы планируете купить новый ноутбук, убедитесь, что на нем есть физические клавиши F1, F3, F4... (например, на некоторых новых компьютерах Mac вместо F-клавиш есть сенсорные панели).

СОЗДАНИЕ ФАЙЛА

Один программист из нашей компании написал самый маленький в мире вирус, он занимал 0 байт, и даже его создатель не знал, что он делает (шутка).

Комбинация для создания файла в Far и MC (Midnight Commander) – Shift+F4. Разница между двумя менеджерами в том, что первый спросит имя файла в начале (перед созданием), а второй – в конце (перед сохранением). В ОС Linux и MacOS (далее мы будем говорить «линукс-совместимые», хотя это и не всегда правда и говорят «юникс-совместимые», или просто *nix) существует команда для создания пустого файла:

```
$ touch app.rb
```

Команда выше создает пустой файл `app.rb` (если файл уже есть, команда меняет время обновления файла на текущее).

По умолчанию файловый менеджер откроет встроенный редактор, где вам будет предложено ввести любой текст. Команда `touch` редактор не открывает, и если вы выполняете свои действия из консоли, то вам потребуется запустить текстовый редактор самостоятельно. Пока тему текстовых редакторов кода опустим, ведь для простейших программ мощный инструмент не нужен.

В текстовом редакторе введите `puts "hello"` и нажмите `Esc`. Вам будет предложено сохранить файл. Сохранить можно также с помощью `F2` (в редакторах кода это почти всегда `Ctrl+S`).

У вас появилась программа в текущем каталоге среди остальных файлов, но мы забыли создать директорию! Тут можно сделать две вещи – удалить файл `F8`, создать директорию `F7` и повторить то же самое там. Или создать директорию и скопировать, нажав клавишу `F5`, туда наш файл. Копирование производится с одной панели на другую, поэтому на одной панели нужно создать директорию, потом переключиться на соседнюю (`Tab`) и оттуда уже скопировать. Можно было переместить файл, нажав клавишу `F6`, чтобы скопированный файл потом не удалять.



Задание

Разберитесь с созданием, копированием, переносом файлов и директорий. Попробуйте скопировать несколько файлов, предварительно выделив их (`Ins`). Ведь это потребуется нам в дальнейшем, а команды для копирования файлов и директорий из консоли не такие очевидные.

Консольный ниндзя

Новичку на первых порах лучше всего хорошо разобраться с файловым менеджером. Мы бы могли дать курс по консольным командам, но сколько человек бы мы потеряли в бою, если бы нужно было овладеть искусством манипулирования файлами в консоли, прежде чем написать первую программу? Ниже мы разберем основные команды. Не все программисты с опытом с ними знакомы, поэтому запоминать их не стоит, но сделать пометку в книге нужно обязательно. Эти команды могут вам потребоваться через год, два и более лет.

Создать директорию (`make directory`) «one»:

```
$ mkdir one
```

Создать одну директорию «one», в ней другую «two» и в ней третью «three». Без флага `p` (`path`) не обойдешься:

```
$ mkdir -p one/two/three
```

Вывести содержимое файла в вашем терминале (`file.txt` – это имя файла):

```
$ cat file.txt
```

Трюк: существует альтернатива команде `cat` (кошка), которая называется `bat` (летучая мышь). На официальном [сайте](#)¹ говорится, что летучая мышь – это кошка с крыльями «A cat with wings». Требуется установить `bat` перед использованием. Из коробки команда позволяет выводить файлы с подсветкой синтаксиса и номерами строк.

Обычно вывод файла осуществляется другой командой, ведь файл может быть большой. Вывести первые 10 строк на экран:

```
$ head -10 file.txt
```

Вывести последние 10 строк на экран:

```
$ tail -10 file.txt
```

Иногда существует какой-то большой текстовый файл, в который постоянно добавляются данные. И вы хотите выводить на экран обновления без перезапуска команды `tail`. В этом случае поможет флаг `f` (`follow` – следовать):

```
$ tail -f file.txt
```

Выход из этой команды осуществляется стандартной комбинацией `Ctrl+C`. Для переименования файла используется команда `mv` (в файл-менеджере F6), от слова `move`. Для компьютера переименовать и переместить файл – это одно и то же. Дело в том, что в таблице размещения файлов (практически в любой стандартной файловой системе) содержатся только структуры с метаданными о файле (имя, размер, атрибуты и т.д.). Содержимое размещено на диске. При переносе или переименовании мы изменяем только таблицу, хотя содержимое остается на том же месте. Именно поэтому перенос больших файлов (гигабайты) занимает доли секунды, если операция выполняется на том же диске. И минуты и часы, когда операция выполняется на разных дисках – ведь нужно «перенести» (на самом деле скопировать и удалить) содержимое. Переименовать первый файл во второй:

```
$ mv file1.txt file2.txt
```

Скопировать файл (сору):

```
$ cp file1.txt file2.txt
```

Скопировать файл в директорию (попробуйте самостоятельно перенести, `move`, файл в директорию):

```
$ cp file1.txt my_directory
```

Переместить файл в домашний каталог:

```
$ mv file.txt ~
```

Скопировать файл в директорию на 1 уровень выше:

¹ <https://github.com/sharkdp/bat>.

```
$ cp file1.txt ..
```

Скопировать файл в директорию на 2 уровня выше (то же самое можно сделать и в файл-менеджере, если указать в качестве назначения директорию `../..`):

```
$ cp file1.txt ../..
```

Скопировать несколько файлов в директорию. К слову, тут уже у многих т.н. высокомерных программистов, которые любят давать советы, наступает клин. Можете использовать этот вопрос «для проверки» – «а знаешь ли ты, какой командой можно скопировать несколько файлов в директорию?»:

```
$ cp {file1.txt,file2.txt} my_dir
```

В Far Manager для копирования нескольких файлов необходимо их сначала выбрать. Это можно сделать с помощью клавиши *Insert* (Ins). Если клавиши *Insert* на вашем компьютере нет (существует только на расширенных клавиатурах), то выбрать можно с помощью `Shift+«стрелка вверх»` или `Shift+«стрелка вниз»`. После этого для копирования с одной панели на другую нажать `F5`.

Если вы установили «*Oh My Zsh*»¹ вместо `bash`, то у вас доступна клавиша *Tab*, которая очень помогает набирать имена файлов. Например, вводите `cp {f,` а потом *Tab*, и оболочка предложит список файлов, которые можно включить в команду. Ничего вводить с клавиатуры не нужно. Очень полезно, когда имена файлов длинные.



Упражнение 1

Откройте свой терминал. Выведите на экран список всех файлов (`ls -lah`). Создайте каталог с именем `my_directory`. Снова выведите список всех файлов, убедитесь, что каталог существует. Выберите любой файл из текущего каталога и скопируйте этот файл в каталог, который вы только что создали. Используйте файловый менеджер, чтобы убедиться, что вы все сделали правильно.

Поиск файла по имени (команда найдет все файлы и директории с расширением `rb`):

```
$ find . -name '*.rb'
```

Поиск всех файлов в текущей директории, в имени которых содержится строка

```
bla:
```

```
$ find . -name '*bla*'
```

Поиск файлов (без директорий) с расширением `rb`:

```
$ find . -name '*.rb' -type f
```

¹ <https://ohmyzsh/>.



Имейте это в виду

Часто люди делают ошибку и ставят два дефиса -- вместо одного - для команды `find`. Например, параметр с двумя дефисами `--name` или `--type f` неверен. Вы должны использовать **один** дефис с `find`. Однако некоторые другие команды Linux принимают два дефиса. Не запутайтесь!

Как вы могли заметить, существуют разные способы поиска файлов в *текущей директории*. Текущая директория обозначается точкой. Двумя точками обозначается директория уровнем выше. Директория двумя уровнями выше обозначается как `../..`. Небольшая справка по разным обозначениям и примеры использования `find`:

- `.` – текущая директория. Пример команды (ищет все файлы с расширением `log` в текущей директории):

```
$ find . -name '*.log'
```

- `..` – директория уровнем выше. Пример команды (ищет все файлы с расширением `log` в директории уровнем выше):

```
$ find .. -name '*.log'
```

- `../..` – директория двумя уровнями выше. Пример команды (ищет все файлы с расширением `log` в директории уровнем выше):

```
$ find ../.. -name '*.log'
```

- `~` – домашняя (home) директория, т.е. личная директория текущего пользователя. Пример команды (ищет все файлы с расширением `log` в домашней директории):

```
$ find ~ -name '*.log'
```

- `/` – корневая (root) директория. Пример команды (ищет все файлы с расширением `log` в корневой директории):

```
$ find / -name '*.log'
```



Упражнение 2

Попробуйте найти все файлы журналов в корневом каталоге.

В Far Manager можно искать файлы с помощью специального диалога, который можно вызвать комбинацией `Alt+F7`. Визуально этот диалог более наглядный, и с ним проще работать. По умолчанию маска файла задана как `*.*` (все файлы, по аналогии с `*.log`, – файлы с расширением `log`). В этом диалоге можно также искать файлы с определенной строкой (например, когда требуется найти все файлы, в которых встречается ваше имя).

Поиск по всем файлам определенной строки (в нашем случае `something`):

```
$ find . -name '*.rb' -type f | xargs grep something
```

Команда выше делает поиск, а потом перенаправляет результат в команду `xargs`, которая для каждой полученной строки запускает программу `grep` с аргументами: `grep something file1.rb`. Не стоит переживать, если эта конструкция не понятна – со временем все встанет на свои места.

Иногда полезно что-то быстро сохранить в файл прямо из консоли. Когда ввод окончен, нужно нажать `Ctrl+D`.

```
$ cat > file.txt
```



Будьте осторожны

Команда `cat > file.txt` затрет предыдущее содержимое файла.

Добавить в конец файла:

```
$ cat >> file.txt
```

Немного про саму файловую систему. Корневой каталог обозначается как `/`. Есть также такое понятие, как «домашний каталог» – это личный каталог текущего пользователя. Узнать имя текущего пользователя можно с помощью команды «кто я»:

```
$ whoami  
ninja
```

Любопытно, что в [Pry](#)¹ (отладчик/дебаггер и REPL, рассматривается ниже) есть команда `whereami` (где я). Она показывает, где вы находитесь в текущем коде (разбирается далее в книге).

Вывести текущую директорию на экран (`PWD` – Print Working Directory – напечатать рабочую директорию):

```
$ pwd  
/home/ninja
```

Домашний каталог обозначается тильдой `~`. Можно вывести его на экран:

```
$ echo ~  
/home/ninja
```

Или совершить другие манипуляции. Создать директорию `tmp` в домашнем каталоге:

```
$ mkdir ~/tmp
```

Скопировать файл в созданную директорию:

¹ <https://github.com/deivid-rodriguez/pry-byebug>.

```
$ cp file.txt ~/tmp
```

Кстати, создайте директорию `~/tmp` – это удобно для хранения временных файлов. Существует системная директория `/tmp`, но все данные оттуда удаляются после перезапуска компьютера (по умолчанию).

Удаление файла, будьте осторожны (`remove`):

```
$ rm file.txt
```

Удаление директории:

```
$ rm -r my_dir
```

Надо заметить, что параметр `r` универсальный для многих команд – он указывает на то, что работа будет производиться с директорией, рекурсивно (`recursive`).



Не делайте этого

Будьте осторожны с командой `«rf»`. Существует самая опасная команда, которую вы можете ввести: `rm -rf /`. Эта команда удалит содержимое корневой директории на вашем диске без какого-либо подтверждения. Иногда в Сети существуют злые шутники, которые могут попросить вас что-нибудь ввести. Всегда проверяйте, что именно вы вводите.

Выше мы рассмотрели команду копирования, но есть еще одна, менее известная, команда копирования, которая вам может пригодиться: `scp`. Эта команда копирует файлы с удаленного сервера на локальный компьютер и обратно. Например, на вашем сайте произошла какая-то ошибка и вы хотите скачать файл с описанием ошибок через SSH-доступ. Это можно сделать с помощью `«scp»`. Остановливайтесь подробно пока на этом не будем, при желании вы всегда можете найти справку в Интернете.

На этом тренировка для настоящих ниндзя окончена, время выпить чаю да съесть еще этих французских булок.

ТЕКСТОВЫЕ РЕДАКТОРЫ

Существует много текстовых редакторов, но мы будем говорить только про редакторы кода. Они отличаются от текстовых редакторов тем, что редакторы типа Word сохраняют файлы не в *plain* (чистом) формате. Нам нужен текстовый редактор, который позволит сохранять файлы *as is* (так, как они есть, ну или почти): т.е. если мы вводим 1 символ и нажимаем «Сохранить», то размер файла будет ровно 1 байт. Если редактор очень простой, то он может быть отнесен как к текстовым, так и к редакторам кода.

Все редакторы кода можно разделить на два вида: консольные и графические. Самый простой консольный редактор – это `nano`:

```
$ nano app.rb
```




```

[ New File ]
^G Get Help   ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell

```

Редактор Nano, работает прямо в вашем терминале

Подсказка внизу – это основные команды. Существуют и другие, более продвинутые редакторы (vim, emacs). К сожалению, для овладения консольными инструментами требуется больше времени. Существует множество холиваров (*holy wars* – святые войны) на тему редакторов кода. Авторы пришли к выводу, что не стоит придавать выбору редактора очень большое значение, так как редактор сам по себе не имеет смысла без наличия знаний по программированию.

Из графических редакторов для Ruby следует выделить четыре (в порядке предпочтений авторов):

- [VsCode](https://code.visualstudio.com/)¹, также известный как Visual Studio Code (не путайте со средой разработки Visual Studio);
- [RubyMine](https://www.jetbrains.com/ruby/)² (платный);
- [Atom](https://atom.io/)³;
- [Sublime Text](https://www.sublimetext.com/)⁴ (платный).

RubyMine относится не к редактору, а к IDE – Interactive Development Environment, это улучшенная версия редактора кода, которую называют «*среда разработки*». Начинающему можно порекомендовать любой из вышеперечисленных. Возможно, кому-то понравится RubyMine, в котором наличие

¹ <https://code.visualstudio.com/>.

² <https://www.jetbrains.com/ruby/>.

³ <https://atom.io/>.

⁴ <https://www.sublimetext.com/>.

широких возможностей облегчает отладку и написание программ, особенно на первых порах. Однако в этой книге работа с тем или иным редактором рассматриваться не будет. Вначале мы будем использовать редактор, встроенный в ваш файловый менеджер (Shift+F4), а в дальнейшем выбор редактора будет только за вами.

Обычно любой редактор при установке создает команду для своего запуска из консоли. С помощью этой команды можно открыть редактор для текущей директории:

```
$ code . # откроем VsCode
$ code ~/tmp # откроем VsCode для каталога tmp
```

Или Atom:

```
$ atom .
```

Если запустить команду без точки, то откроется каталог по умолчанию. На практике редко приходится запускать редактор без параметра.



Задание

Установить текстовый редактор. Попробовать создать несколько файлов в текстовом редакторе. В каждый файл запишите имя человека, которого вы знаете. Удалите файлы.

ПЕРВАЯ ПРОГРАММА

На самом деле нашей первой программой была программа сложения двух чисел: `puts 1+1`. Давайте создадим новый файл с именем `app.rb` и запишем в него следующий код:

Ваша первая программа

```
puts "I would hug you, but I'm just a text"
```

Когда файл создан и сохранен, из терминала можно запустить программу:

```
$ ruby app.rb
I would hug you, but I'm just a text
```

В файл-менеджере тоже можно ввести `ruby app.rb`. Но что такое? Если запустить программу через файл-менеджер, то все пропадет! Тонкость в том, что программа запускается, «отрабатывает» и управление переходит обратно – в терминал или в нашем случае в файловый менеджер. Поэтому чтобы посмотреть, «что же там было» после того, как мы нажали Enter, надо нажать Ctrl+O.

Ура! У нас получилась первая осмысленная программа. Давайте ее немного улучшим:

Выведите текст и дождитесь клавиши Enter

```
puts "I would hug you, but I'm just a text"
gets
```

Теперь мы выводим на экран строку и вместо того, чтобы выходить из программы, ожидаем ввода. Но не просто ввода, а ввода строки. Инструкция `gets` – это по сути `get string` – получить строку. Вот мы и пробуем получить строку. Заметьте, что строка может состоять из множества символов, поэтому Ruby понимает окончание строки только в том случае, если вы нажмете `Enter`. Разумеется, можно просто нажать `Enter`, тогда строка будет пустая (если честно, то не совсем, но будет «казаться», что она пустая).

Запустите программу выше и попробуйте нажать `Enter`. Если вы запускаете программу из файл-менеджера, то результат не «пропадет» и программа будет ждать вашего ввода.

Давайте составим простейшую программу для изучения иностранного языка. Возьмем три слова: `ball`, `door`, `peace`. Представим, что нам нужно выучить эти слова. Мы напишем программу, которая будет спрашивать «Как переводится слово `peace`?». В этот момент подразумевается, что пользователь должен дать ответ вслух: мяч, дверь, мир. Так как с остальными операторами языка мы незнакомы, то обойдемся тем, что есть:

Программа обучения иностранному языку

```
1 puts "How to translate ball?"
2 gets
3 puts "How to translate door?"
4 gets
5 puts "How to translate peace?"
6 gets
```

Попробуем запустить – работает! Это не очень удобное, но рабочее и полезное приложение. Оно не выводит ответы, но уже задает вопросы. Другими словами, с помощью двух операторов `put` и `gets` мы смогли написать что-то интересное. Что же будет дальше! Для играющих на гитаре предлагаем программу для изучения нот на первой струне:

Программа обучения игре на гитаре

```
1 puts "Say a note on a 0 fret?" # Ответ E
2 gets
3 puts "Say a note on a 1st fret?" # Ответ F
4 gets
5 puts "Say a note on a 2nd fret?" # Ответ F#
6 gets
7 puts "Say a note on a 3rd fret?" # G
```

```
8 gets
9 # ...
```

И так далее, до 12-го лада (E F F# G G# A A# B C C# D D# E). Напишите программу самостоятельно. Если тема музыки вам не интересна, сделайте программу для изучения 10 слов.

По поводу листинга выше можно сделать несколько замечаний. Во-первых, вы, наверное, уже заметили, что после строки можно оставить любой комментарий, достаточно ввести # (решетка, hash, иногда говорят *round sign*). Можно оставлять комментарий и на новой строке. Можно оставлять сколько угодно комментариев и пустых строк, на работу программы это не влияет.



Упражнение 1

Попробуйте оставить комментарии к своей программе и добавить пустые строки после `gets`, чтобы визуально программа выглядела «легче».

Второе замечание – поддержка русского языка, а точнее правильной кодировки. В ОС Windows, скорее всего, возникнут проблемы с русской кодировкой. Это одна из причин, почему не стоит использовать Windows и нужно переходить на MacOS или Linux – на этих операционных системах проблем с кодировкой нет. К счастью, проблема кодировки очень просто исправляется, если в самое начало файла добавить:

```
# encoding: cp866
```

Разумеется, файл должен быть тоже сохранен в этой кодировке в текстовом редакторе. Другими словами, мы «дружим» – Ruby и текстовый редактор. Интерпретатору Ruby говорим, в какой кодировке будет этот файл, а в редакторе выбираем эту самую кодировку CP866 (также она может называться DOS-кодировкой). После этого можно писать по-русски.

В «нормальных» операционных системах этих трюков проделывать не нужно. Если можете, переключайтесь на них как можно скорее. В дальнейшем таких сложных трюков быть не должно, но помните: если что-то не получается, то ошибка может заключаться в том, что вы используете неправильную операционную систему. Несмотря на то что Ruby должен без проблем работать в Windows, для этой операционной системы он не предназначался. А авторы популярных библиотек не тестируют свои программы на Windows.



Упражнение 2

Если у вас установлена ОС Windows, попробуйте скачать VMWare Workstation (платная программа) или [VirtualBox](https://www.virtualbox.org/)¹ (бесплатная). Это виртуальная машина – программа для запуска операционных систем внутри вашей ОС. Попробуйте запустить виртуальную машину и уста-

¹ <https://www.virtualbox.org/>.

новить в ней [Linux Mint Cinnamon edition](#)¹. Попробуйте написать первую программу в Linux! Если не получится – ничего страшного, продолжайте обучение дальше, можно будет вернуться к этому позднее.

ПЕРЕМЕННЫЕ В ЯЗЫКЕ RUBY

Переменная – это область памяти в компьютере, куда мы можем сохранить значение во время исполнения программы. Возникает вопрос: а зачем его сохранять? А как раз для того, чтобы его потом изменить. В этом и заключается суть переменных – это ячейки памяти, куда мы можем что-то записать и, при желании, изменить.

Но не обязательно менять значения переменных, можно создавать переменные для удобства. Правда, в этом случае переменные часто называют константами – ведь они не меняются! Поэтому в современном языке JavaScript для создания переменных есть два ключевых слова: *let* для создания переменной и *const* для создания константы. Но в Ruby все проще.

Попробуем «объявить» (создать, *define, declare, create, make*) простую переменную:

```
1 puts "Your age?"
2 age = gets
3 puts "Your age is"
4 puts age
```

В программе выше мы спрашиваем возраст. После того как возраст указан, программа выведет на экран ответ:

```
Your age?
20
Your age is
20
```

Возраст, который мы вводим, сохраняется в переменную *age*. Мы бы могли назвать ее другим именем (например, *a*), но в этом случае и на четвертой строке пришлось бы писать «puts a». Существуют т.н. *naming conventions* – соглашения о наименовании, их достаточно просто найти: ввести в поисковой системе запрос «*naming conventions variables ruby*».

В языках программирования Ruby и JavaScript мы столкнемся с тремя основными *naming conventions*:

- Snake case (snake – змея), между словами ставится знак подчеркивания underscore (`_`). Переменные именуются следующим образом:

```
client_age
user_password
user_password_expiration_date
```

¹ <https://linuxmint.com/download.php>.

Используется в Ruby, а также в базах данных;

- Camel case (camel – верблюд), слово начинается с маленькой буквы, слова разделяются с помощью больших букв:

```
clientAge
userPassword
userPasswordExpirationDate
```

Используется в JavaScript;

- Kebab case (kebab – шашлык), слова разделяются дефисом:

```
client-age
user-password
user-password-expiration-date
```

Иногда используется в HTML, в т.н. data-атрибутах. Например:

```
<input type="text" name="login" data-error-highlight-color="red">
```

Пока запомним только первый вариант, для Ruby. Если переменная имеет длинное название, то слова разделяем нижним подчеркиванием. Нужно заметить, что чем короче названия переменных, тем лучше. Всегда нужно стремиться писать код так, чтобы названия переменных не были слишком длинными. Однако для начинающих эта задача не всегда по силам. Придумать хорошее название для переменной не всегда просто, среди программистов ходит даже такая шутка:

There are only two hard things in Computer Science: cache invalidation and naming things.

Дословно: существуют две сложные проблемы в Компьютерной Науке: инвалидация кеша и именование вещей.

Если название переменной получается слишком длинным, не стоит его «искусственно» занижать (например, переименовав `client_password_expiration_date` в `cred`). Обычно это свидетельство того, что контекст решаемой проблемы слишком широкий и пришла пора разбить функциональность на малозависимые друг от друга классы/объекты. Однако это задача для другой книги. На данном этапе можете называть переменные так, как вам хочется.

Кроме `naming conventions`, существуют правила: в Ruby переменные должны начинаться всегда с буквы, переменные могут содержать цифры и/или знак подчеркивания.



Задание

Написать программу, которая подряд спрашивает год рождения, место рождения, номер телефона трех клиентов, после чего выводит полученную информацию полностью в виде «карточек» (в англ. языке это бы называлось `baseball card`, аналогия в русском языке – карточка из картотеки).

Так как придумать названия на первом этапе на английском языке может быть сложно (а для переменных желательно, но не обязательно использовать английский язык, а не транслит), приведем перевод. Год рождения – year of birth, место рождения – place of birth, телефонный номер – phone number. На будущее при возникновении вопросов об именовании переменных рекомендуется заглянуть в словарь:

[Русско-английский и англо-русский словарь](#)¹

[Альтернативный словарь с контекстом](#)²

[Поиск синонимов англ. языка](#)³

[Переводчик на все случаи жизни](#)⁴

СЛОЖЕНИЕ И УМНОЖЕНИЕ СТРОК

Давайте посмотрим на нашу программу, что мы можем в ней улучшить?

```
1 puts "Your age?"
2 age = gets
3 puts "Your age is"
4 puts age
```

Две последние строки можно сократить до одной:

```
1 puts "Your age?"
2 age = gets
3 puts "Your age is" + age
```

Результат работы программы:

```
Your age?
30
Your age is30
```

Чего-то не хватает? Правильно, пробела после слова «is». Как вы уже увидели из примера выше, мы можем складывать строки. С точки зрения математики это не имеет никакого смысла, зато строки в памяти компьютера объединяются. Запустите такой код в REPL или в виде программы:

```
"My name is " + "Roman" + " and my age is " + "30"
```

Результат:

```
"My name is Roman and my age is 30"
```

¹ <https://www.multitran.ru/>.

² <http://context.reverso.net/>.

³ <http://www.thesaurus.com/>.

⁴ <https://translate.google.com/>.

Попробуйте теперь сложить два числа в виде строк следующим образом, постарайтесь понять, каким будет ответ:

```
"100" + "500"
```

Спойлер: ответ будет "100500". Другими словами, если число представлено в виде строки (взято в кавычки), Ruby будет понимать это число как строку. Если мы напишем `100 + 500` (не берем в двойные кавычки каждое число), то результат будет 600.

Оказывается, что строки можно не только складывать, но и умножать. Только строку нужно умножать на число, в примере ниже нельзя взять второе число в кавычки:

```
"10" * 5
=> "1010101010"
```

Получили число "10", повторенное 5 раз. Если мы поставим после "10 " пробел, результат будет более наглядным:

```
"10 " * 5
=> "10 10 10 10 10 "
```

Как было уже замечено, "10 " – это всего лишь строка, можно подставить любую строку:

```
"Я молодец! " * 10
=> "Я молодец! Я молодец! Я молодец! Я молодец! Я молодец! Я молодец! \
Я молодец! Я молодец! Я молодец! Я молодец! "
```

На практике приходится часто умножать "=" или "-" на 80 (ширина экрана в символах, принятая за стандарт), чтобы визуально отличить одну часть от другой. Например:

```
puts "Your age?"
age = gets
puts "=" * 80
puts "Your age is " + age
```

Результат:

```
Your age?
```

```
30
```

```
=====
```

```
Your age is 30
```