Оглавление

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 Ру́би (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 нескучным.

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 работ[1], 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 чате[1].

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, подумать 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 «OK», 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.

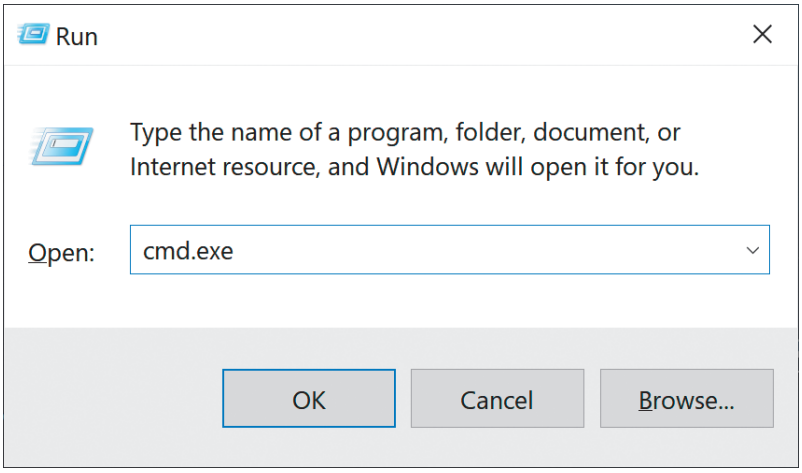
Настройка Windows для запуска первой программы

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://ohmyz.sh/.

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 вверху 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 завязываться 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.

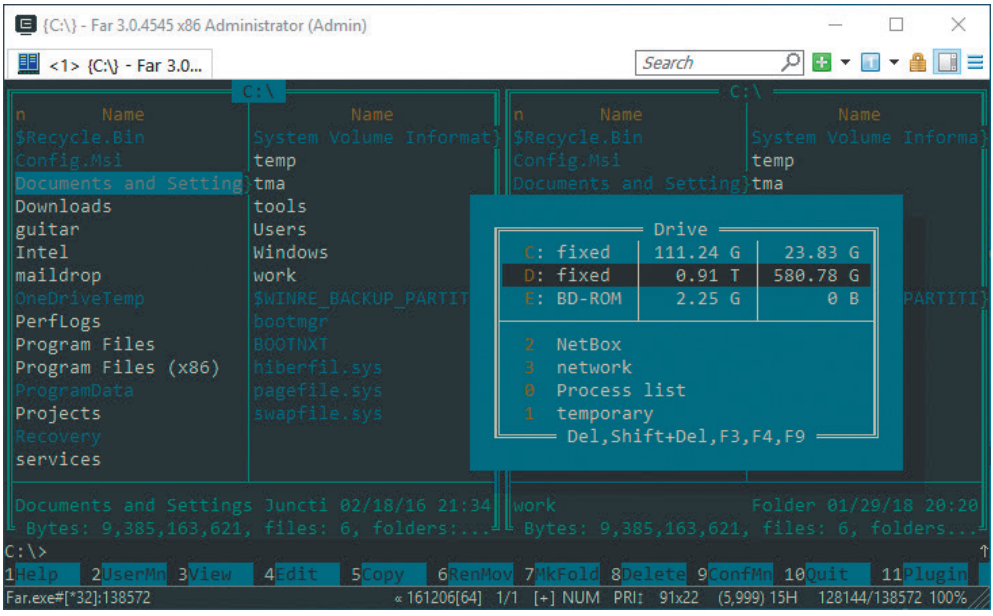
[1] http://pry.github.io/.

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:

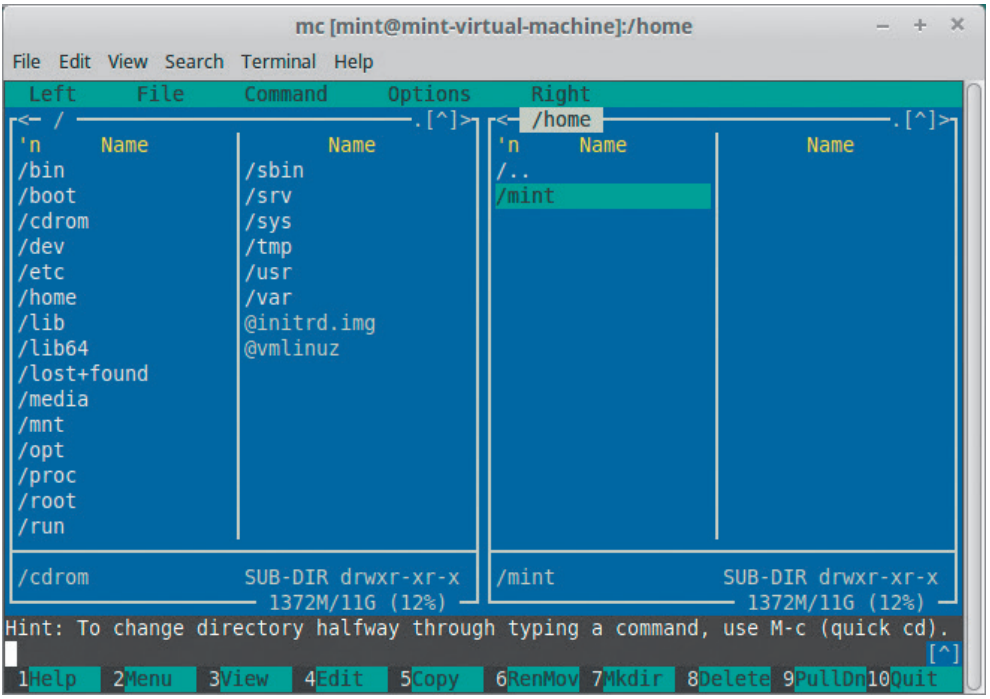


Far Manager запущен на Windows

Работа с файлами – ключевой навык программиста, сисадмина или даже любого эникейщика. 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'а 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.