ruby_on_rails_tutorial

Rails tutorial logo

Welcome, 최연호(R 팀) | 목차 | Log Out

Chapter 1 시작에서 배포까지

루비온 레일스 튜토리얼: 레일스로 웹 개발 배우기에 온 것을 환영한다. 이 책의 목표는 독자들이 인기 있는 웹 프레임워크인 루비온레일스를 이용하여 웹 애플리케이션을 개발할 수 있게 하는 것이다. 웹 개발이 처음이라면, 루비온레일스 튜토리얼을 통해서 루비, 레일스, HTML 과 CSS, 데이터베이스, 소스버전 관리, 테스팅, 개발에 대한 기초교육을 포함하는 웹 애플리케이션 개발에 대한 전반적인 내용을 소개받을 수 있다. 반대로 이미 웹 개발에 익숙한 독자들은 MVC, REST, 제네레이터, 마이그레이션, 라우팅, ERB 등, 루비온레일스의 주요 기능들을 빠르게 학습 할 수 있다. 어떤 경우라도, 루비온레일스 튜토리얼을 학습하고 나면, 도서, 블로그, 스크린캐스트 등의 교육 시스템에서 다루는 심화 과정을 좀 더 잘 이해 할 수 있게 될 것이다.1http://www.railstutorial.org/. 이 책을 오프라인에서 읽고 있다면, http://www.railstutorial.org/book 에서 레일스 튜토리얼의 온라인 버전의 최신 업데이트를 확인 할 수 있다.

루비온레일스 튜토리얼은 세 개의 예제 애플리케이션을 점진적으로 세련되게 만들어 가면서 웹 개발을 통합적으로 접근한다. 처음에는 최소한의 기능만을 가지는 hello 앱 (섹션 1.3) 으로 시작해서 조금 더 발전 된 toy 앱 (2 장)을 만든 후 실제 sample 앱 (3 장에서 12 장 까지)을 만들어 볼 것이다. 일반적인 단어로 프로그램 이름을 정한 것은 루비온레일스 튜토리얼 에서 개발하는 애플리케이션이 어떤 특정 분야의 웹사이트 개발에만 사용되는 것이 아니란 의미를 내포 하고 있다. 비록 최종적으로 만들게 될 샘플 애플리케이션은 어느 인기 있는 소셜 마이크로 블로깅 사이트 (이 사이트는 우연하게도 처음 개발될 때 레일스로 작성되었다)와 매우 비슷하지만, 이 튜토리얼에서 강조하는 것은 기본 원리를 깨우쳐야만 독자들이 만들고자 하는 어떠한 종류의 웹 애플리케이션도 만들 수 있다는 것이다.

흔히 접하게 되는 질문은 루비온레일스 튜토리얼로 웹 개발을 학습할 때 배경 지식이 어느 정도 필요한가다. 섹션 1.1.1 에서 좀 더 깊게 다루겠지만, 웹 개발은 초보 개발자들에겐 쉽지 않은 분야다. 처음 이 튜토리얼을 작성할 때에는 프로그램 개발 경험이 있거나 웹 개발 경험이 있는 독자를 대상으로 했다. 그런데 상당수의 독자가 소프트웨어 개발을 시작하는 단계라는 것을 알게 되었다. 그래서 이번 레일스 튜토리얼 3 판에서는 레일스의 진입장벽을 낮추기 위해 몇 가지 큰 변화를 주었다. (글상자 1.1).

글상자 1.1. 진 입장벽 낮추기

루비온레일스 튜토리얼 3 판에서는 레일스 개발의 진입 장벽을 낮추기 위해 다음과 같이 진행하였다.

- 클라우드에서 개발하는 표준 개발 환경을 사용하였다.(섹션 1.2), 이를 통해 서버 초기 설치 및 설정 과정의 부담을 줄였다.
- 레일스에 내장되어 있는 테스트 프레임워크인 MiniTest 를 포함한 레일스의 “기본 구성”을 사용해 개발하였다.
- 다수의 외부 라이브러리 의존성(RSpec, Cucumber, Capybara, Factory Girl)을 제거하였다.
- 테스팅할 때는 가볍고 유연한 방식으로 접근했다.
- 복잡한 설정(Spork, RubyTest)에 대한 내용은 뒤로 미루거나 삭제했다.
- 특정 레일스 버전에 국한된 기능 보다는 웹 개발의 일반적인 원칙을 더 강조했다.

이와 같은 변화로 더 많은 독자들이 이전 버전보다 루비온레일스 튜토리얼 3 판을 쉽게 읽기를 기대한다.

첫 장에서는 루비온레일스 개발을 시작하기 위해 필수 소프트웨어를 설치하고 개발 환경을 구성할 것이다(섹션 1.2). 그리고 첫번째 레일스 애플리케이션으로 hello_app 을 만들 것이다. 레일스 튜토리얼은 일반적인 소프트웨어 개발 관례를 따르기 때문에 레일스 프로젝트를 생성한 직후 바로 Git 버전 관리 시스템에 등록 할 것이다 (섹션 1.4). 그 다음, 믿기 어렵겠지만 첫 번째 장에서부터 개발한 첫번째 앱을 운영 환경으로 배포할 것이다(섹션 1.5).

2 장 에서는, 두번째 프로젝트를 만들어 레일스 애플리케이션이 어떻게 동작하는지 보여줄 것이다. 빠른 진행을 위해 지저분하고 복잡한 코드 대신 스캐폴드 기능을 이용해 (글상자 1.2) toy 앱 (말 그대로 toy_app 으로 부르겠다) 코드를 생성하겠다. 2 장에서는 URI (보통 URL 이라 불리는)2 로 브라우저와 toy 앱의 상호 작용에 대해 알아보겠다.

튜토리얼의 나머지에서는 모든 코드를 처음부터 작성하여 하나의 거대한 실제 sample 애플리케이션(sample_app 이라 부르겠다)을 개발하는데 집중할 것이다. 목업(mockup), 테스트 주도 개발(TDD), 통합 테스트을 조합하여 sample 애플리케션을 개발할 것이다. 3 장 부터는 정적 페이지를 만든 후 동적 내용을 추가할 것이다. 그리고 4 장 에서는 레일스의 바탕이 되는 루비 언어에 대해 간단히 살펴 볼 것이다. 그 후 5 장 부터 10 장 까지, sample 애플리케이션의 기초를 잡을 것이다. 사이트의 레이아웃을 잡고, 사용자 모델을 추가 한 후, 사용자 등록 및 인증 시스템(사용자 계정 활성화 및 비밀번호 초기화를 포함하는)을 추가할 것이다. 마지막으로 11 장 과 12 장 에서는 마이크로 블로그와 소셜 네트워킹 기능을 추가해서 실제로 동작하는 예제 사이트를 만들게 될 것이다.

Box 1.2. 스 캐폴딩: 더 빠르고 더 쉽고 더 매혹적임

애초에 레일스가 인기를 끌게 된 데에는 레일스를 만든 데이비스 하이네마이어 핸슨의 유명한 동영상 블로그 15 분만에 만들기를 통해 개발을 신나게 할 수 있다는 것을 보여 준 것에서 기인한다. 레일스의 강력함을 느껴보고 싶다면 이 동영상과 후속작들을 보라. 미리 말해 두지만, 이들은 15 분 묘기를 위해 스캐폴딩이라는 레일스 기능을 사용했다. 주로 코드를 자동 생성해 주는 스캐폴딩은 레일스의 generate scaffold 명령으로 생성한다.

저자들이 처음 루비온레일스 튜토리얼을 작성할 때는, 스캐폴딩 방법론을 사용하는 유혹을 느꼈다. 스캐폴딩은 더 빠르고, 더 쉽고, 더 매혹적인 기술이다. 하지만 스캐폴딩이 가진 복잡성과 극히 간결한 코드에 초보 레일스 개발자들이 완전히 압도 당할 수 있다. 쉽게 사용할 수 있겠지만, 결코 이해하지 못한 채로 사용하게 될

것이다. 스캐폴딩 방법론을 따라하면 레일스를 잘 이해하지 못한 채 하루 하루 스크립트만 찍어내는 기계가 될 위험부담이 있다.

루비온레일스 튜토리얼은 양 극단의 접근법을 둘 다 사용 할 것이다. 2 장 에서는 스캐폴딩으로 toy 앱을 개발하지만, 레일스 튜토리얼의 핵심인 sample 앱을 개발하는 3 장 부터는 코드를 실제로 작성하기 시작할 것이다. 이 sample 애플리케이션을 개발하는 각 단계에서는 이해 가능한 범위내에서 (너무 길어서 어렵게 느껴지지 않을 정도의) 짧은 길이의 코드를 작성 할 것이다. 이 과정이 쌓이면 레일스를 깊이 이해하고 자유롭게 사용할 수 있게 되고 아마도 어떤 종류의 웹 애플리케이션이든 개발 할 수 있는 배경 지식이 될 것이다.

1.1 소개

루비온레일스(혹은 간단히 "레일스"로 부른다)는 루비 프로그래밍 언어로 작성 된 웹 개발 프레임워크다. 다이내믹 웹 애플리케이션 개발 분야에 있어 루비온레일스는 2004 년에 데뷔하여 단숨에 가장 강력하고 인기 있는 도구로 손꼽히게 되었다. 레일스는 Airbnb, Basecamp, Disney, GitHub, Hulu, Kickstarter, Shopify, Twitter, 그리고 Yellow Pages 같이 다양한 회사에서 사용 중이다. 또한 레일스에 특화된 웹개발 전문회사, 예를 들어 ENTP, thoughtbot, Pivotal Labs, Hashrocket, 그리고 HappyFunCorp 등이 있고, 거기에 셀 수 없이 많은 독립 컨설턴트, 트레이너, 하청 업체에서 레일스를 사용하고 있다.

어떻게 레일스가 이토록 성장하게 되었을까? 무엇보다 루비온레일스는 100% 오픈소스 프로젝트로 사용에 제약이 없는 MIT 라이센스를 따르고 있기 때문에 다운로드 하고 사용하는데 전혀 비용이 들지 않는다. 레일스의 성공 이면에는 루비 언어의 유연성을 이용한 우아하고 간결한 설계를 바탕으로, 웹 애플리케이션 개발에 특화된 도메인 전용 언어(DSL)를 제공해 준다. 이에 따라 웹 프로그래머가 하는 많은 업무(예를 들어, HTML 생성, 데이터 모델 작성, URL 라우팅)를 레일스로 쉽게 수행할 수 있고 그 결과물인 애플리케이션 코드는 간결하고 가독성이 높아졌다.

레일스는 웹 분야의 신기술과 다른 프레임워크 설계를 빠르게 도입했다. 그 예로 레일스는 웹 애플리케이션 구조에 REST 방식 설계를 완벽하게 수용한 최초의 프레임워크 중 하나다(이 부분은 이 튜토리얼을 통해 배우게 될 것이다). 그리고 다른 프레임워크가 새로운 기술을 개발하면 레일스의 창시자 데이비드 하이네마이어 핸슨과 레일스 코어팀은 이를 적극적으로 도입해 왔다. 이를 극명하게 드러내는 예가 레일스와 레일스의 경쟁 웹 프레임워크였던 Merb 의 병합일 것이다. 이로써 레일스는 Merb 의 모듈 설계, 안정된 API 와 성능의 향상 같은 잇점을 누릴수 있게 되었다.

마지막으로 레일스는 다수의 열정적인 커뮤니티를 가지고 있다는 장점이 있다. 여기에는 수 백명의 오픈소스 기여자와 참여도가 높은 컨퍼런스, 엄청난 수의 Gem(페이징이나 이미지 업로드 같은 특정 작업에 대한 솔루션을 내장한 라이브러리)들과 다양한 블로그 및 개발자 포럼, IRC 대화방을 들 수 있다. 레일스 개발자가 늘어나면서 알 수 없는 애플리케이션 에러에 대해 "에러 메시지를 구글링하기" 로 가장 연관성 높은 블로그 글이나 개발자 포럼 글타래를 찾을 수 있게 된 부분도 있다.

1.1.1 전제 조건

이 책의 공식적인 전제 조건은 없다. 루비온레일스 튜토리얼은 레일스만 다루지 않고 루비 언어, 레일스의 기본 테스트 프레임워크(MiniTest), 유닉스 커맨드라인, HTML, CSS, JavaScript, SQL 을 모두 조금씩 다루는 통합된 튜토리얼이다. 이 튜토리얼이 다양한 범위의 내용을 다루고 있지만, 시작하기 전에 HTML 과 프로그래밍에 대한 배경 지식을 갖고 있다면 더욱 좋다. 놀랍게도 꽤 많은 초보자들이 처음부터 루비온레일스 튜토리얼로 웹 개발을 시작하는데, 만약 독자가 이 경우라면, 내가 제시한 대로 해보길 바란다. 너무 어렵게 느껴진다면 언제든지 이전으로 돌아가서 아래에 나온 자료로 시작하면 된다. 많은 독자들이 추천한 또 다른 전략은 이 튜토리얼을 두 번 따라해 보는 것이다. 두 번째 경우에는 처음에 얼마나 많이 배웠는지 되새길 수 있고, 또 이전보다 쉽게 느껴질 것이다.

일반적인 질문 중에 레일스를 학습할 때 루비 언어를 먼저 학습해야 하는 가라는 것이 있다. 그 답은 개인이 선호하는 학습 방식과 프로그래밍 경험이 얼마나 있느냐에 달려있다. 체계적으로 처음부터 학습하길 원하거나, 프로그램 개발 경험이 없다면, 루비 언어부터 학습하는 것이 좋을 것이다. 이 경우에는, 크리스 파인의 Learn to Program 과 피터 쿠퍼의 Beginning Ruby 를 추천한다. 한편 많은 초급 레일스 개발자들은 웹 페이지 하나 만들기 전에 루비 언어 책을 통달하지 않고도 웹 애플리케이션을 만들 수 있다는 것에 놀라지 않을 수 없다. 이 경우에도 레일스 튜토리얼을 시작하기에 앞서 Try Ruby3 에 서 루비 언어 인터랙티브 튜토리얼은 따라 해보는 것이 좋다. 이 튜토리얼이 어렵게 느껴진다면, 루비온레일스 튜토리얼보다 더 초심자를 위해 작성된 대니얼 코허의 Learn Ruby on Rails 나 One Month Rails 를 추천한다.

이 튜토리얼을 마치게 되면 인터넷 상의 수많은 레일스 중/고급 자료를 쉽게 이해하게 될 것이다. 아래는 그 중 추천하는 자료 목록이다.

- Code School: 양질의 대화형 온라인 프로프래밍 교육 과정

- The Turing School of Software & Design: 콜로라도주 덴버에서 진행되는 27 주짜리 풀타임 트레이닝 프로그램. 레일스 튜토리얼 독자는 할인코드 RAILSTUTORIAL500 를 사용할 경우 500 달러 할인 해택을 받을 수 있다.
- Tealeaf Academy: 양질의 온라인 레일스 개발 초급과정(심화 과정 포함)
- Thinkful: 이 튜토리얼과 비슷한 수준의 온라인 수업
- RailsCasts by Ryan Bates: 라이언 베이츠의 최고의 레일스 스크린캐스트(대부분 무료)
- RailsApps: 다양한 주제별 레일스 프로젝트와 튜토리얼
- Rails Guides: 주제 중심, 최신의 레일스 참고서

1.1.2 이 책의 작성 관례

이 책의 작성 관례는 대부분 직관적이다. 여기서는 예외적인 몇가지 사항들을 기술하겠다.

이 책의 많은 예제는 커맨드라인 명령을 사용한다. 간단히 말해, 모든 커맨드라인 예제는 아래와 같이 유닉스 스타일의 커맨드라인 프롬프트(달러 기호)를 사용한다.

```
$ echo "hello, world"
hello, world
```

섹션 1.2 에서 언급했듯이, 클라우드 개발환경 (섹션 1.2.1) 을 사용하는 것을 (특히 윈도우 사용자들에게) 권장하는데, 클라우드 개발환경에는 자체에 유닉스(리눅스) 커맨드라인을 포함하고 있기 때문이다. 이것은 레일스가 커맨드라인에서 실행되는 명령을 많이 내장하고 있기 때문에 매우 유용한 작업환경이 된다. 예를 들어 섹션 1.3.2 에서 rails server 명령으로 로컬 개발서버를 시작한다.

```
$ rails server
```

커맨드라인 프롬프트와 마찬가지로 레일스 튜토리얼은 유닉스 방식의 디렉토리 구분자를 사용한다. (슬래시 문자 /). 그 예로 sample 애플리케이션의 production.rb 설정 파일의 경로는 아래와 같이 표시한다.

```
config/environments/production.rb
```

이 파일 경로는 애플리케이션 루트에 대한 상대 경로라는 것을 숙지해야 한다. 시스템에 따라 이 경로는 바뀔 수 있는데, 클라우드 통합 개발환경(섹션 1.2.1) 에서 절대 경로는 다음과 같다.

```
/home/ubuntu/workspace/sample_app/
```

따라서 production.rb 의 전체 경로는 아래와 같다.

/home/ubuntu/workspace/sample_app/config/environments/production.rb

이후로는 애플리케이션 절대 경로를 생략하여 config/environments/production.rb 처럼 간략하게 기술할 것이다.

레일스 튜토리얼을 진행하면서 여러 프로그램(쉘 명령어, 버전 관리 상태, 루비 프로그램등)에서 출력하는 결과를 보게 될 것이다. 각 컴퓨터 시스템간의 미미한 차이로 이 튜토리얼의 결과와 독자들이 보는 화면이 일치 하지 않을 수 있는데 그리 중요한 사항이 아니므로 무시하겠다. 아마도 독자가 사용하는 시스템에 따라 어떤 명령은 에러가 발생할 수도 있는데, 이 튜토리얼에서는 이런 에러를 시지프스처럼 매번 일일이 설명하는 대신, 실제 소프트웨어 개발에서도 유용하게 사용되는 "에러 메시지 구글링 하기" 방식을 따르는 것으로 대신하겠다. 이후의 튜토리얼을 진행하면서 실행이 잘 안될 때엔 레일스 튜토리얼 도움말 섹션 에서 자료를 찾아보면 좋을 것이다.4

레일스 튜토리얼은 레일스 애플리케이션의 테스트와 관련된 내용도 다루고 있는데, 이는 어떤 코드가 테스트에 성공(녹색으로 표현된다)하고 실패(빨간색으로 표현된다) 하는지 아는 게 도움이 되기 때문이다. 편리하게도 테스트에서 실패한 코드는 빨간색으로 표시되고, 테스트를 통과한 코드는 녹색으로 표시된다.

각 장에는 연습문제가 있는데, 독자의 재량에 달려 있지만 가능하면 문제를 풀어 보는 것이 좋다. 주제를 벗어나지 않도록 연습문제의 답은 그 이후의 코드 목록에는 포함되지 않게 할 것이다. 드물게 연습문제 중 이후에 다룰 코드를 사용하는 경우가 있는데, 이런 경우엔 본문에서 별도로 다룬 경우에만 나올 것이다.

마지막으로 루비온레일스 튜토리얼은 코드 예제를 좀더 쉽게 이해할 수 있도록 편의상 두 가지 관례를 따르고 있다. 우선 코드를 기술할 때 아래처럼 한 줄 혹은 그 이상으로 굵게 강조 된 경우다.

```
class User < ActiveRecord::Base
  validates :name,  presence: true
  validates :email, presence: true
end
```

이렇게 강조된 줄은 해당 예제에서 가장 중요한 코드라는 것을 표시한다. 또 이전 예제에서 달라진 부분을 표시하기도 한다. 두번째로, 간결함과 단순성을 위해 이 책의 코드 예제에는 아래와 같이 수직으로 배열된 점을 포함한다.

```
class User < ActiveRecord::Base
  .
  .
  .
  has_secure_password
end
```

이 점들은 코드가 생략되었다는 뜻이므로 이것까지 복사해서는 안된다.

1.2 기동과 실행

경험 많은 레일스 개발자에게도 루비나 레일스, 그외의 필요한 소프트웨어를 설치하는 것은 고된 일이다. 설치 과정에서 발생하는 문제들이 서로 다른 운영체계, 버전 차이, 텍스트 에디터나 통합 개발환경(IDE)의 설정 차이 등이 복합적으로 영향을 미치기 때문이다. 이미 자신의 머신에 개발환경을 구성한 독자라면 지금의 환경을 그대로 사용해도 무방하지만, (글상자 1.1 에 도 애기했듯이) 레일스를 처음 접하는 독자들은 클라우드 통합 개발환경을 사용하길 바란다. 클라우드 통합 개발환경은 일반 웹브라우저에서 동작하고 서로 다른 플랫폼에서도 동일하게 동작한다. 이는 전통적으로 레일스 개발이 힘들다고 알려진 운영체계(윈도우)를 사용할 때 특히 유용하다. 그럼에도 불구하고 루비온레일스 튜토리얼을 윈도 로컬 개발환경에서 진행하고자 한다면 InstallRails.com 의 가이드를 따르도록 한다.5

1.2.1 개발 환경

개발 환경을 구성하는 방법에는 전 세계 레일스 개발자 수 만큼 여러가지 방법이 있을 것이다. 이로 인한 혼동을 피하기 위해 루비온레일스 튜토리얼은 클라우드 개발 환경인 Cloud9 을 기준으로 작성되었다. 특별히 이번 3 판에서는 Cloud9 에서 이 튜토리얼을 진행하는데 적합한 개발환경을 협찬해줘서 기쁘게 생각한다. Cloud9 의 레일스 튜토리얼 워크스페이스는 루비언어, 루비젬, Git 같은 프로 수준의 레일스 개발 환경이 미리 설정되어 제공된다(사실 따로 설치할 소프트웨어 중 가장 큰 것은 레일스 자체이지만 이는 의도된 바다.(섹션 1.2.2)) 클라우드 통합개발환경에는 웹 애플리케이션을 개발하는데 필수적인 텍스트 에디터, 파일 시스템 탐색기, 커맨드라인 터미널도 제공하고 있다. (그림 1.1). 다른 기능 중에서도 클라우드 통합 개발환경의 텍스트 편집기는 대규모 루비온레일스 프로젝트에서 코드를 검색하는데 필수적인 “Find in Files” 기능을 제공하여 6 마지막으로 클라우드 통합 개발환경를 실제 개발에는 사용하지 않는 경우에도 텍스트 편집기나 다른 개발툴의 일반적인 기능들에 대한 훌륭한 경험이 될 것이다.

images/figures/ide_anatomy
그림 1.1: 클라우드 통합 개발환경 구조.

클라우드 개발환경을 사용하려면 다음의 단계를 밟는다.

1. Cloud9 에서 무료 회원 가입하고 7
2. “Go to your Dashboard”를 클릭한다.
3. “Create New Workspace”를 선택하고
4. 그림 1.2 과 같이, “rails-tutorial”이란 워크스페이스를 만든다 (“rails_tutorial”이 아니다), “Private to the people I invite”로 설정하고, Rails Tutorial 아이콘을 선택한 후 (루비온레일스 아이콘이 아니다)
5. “Create”를 클릭한다.
6. Cloud9 가 워크스페이스를 프로비저닝하고 나면 이를 선택하고 “Start editing”을 클릭한다.

두개의 공백문자를 사용하는 것이 루비 언어의 들여쓰기 규칙 상의 사실상 표준이므로, 편집기의 설정 역시 네개의 공백문자에서 두개로 변경하는 것을 권장한다. 그림 1.3 과 같이, 오른쪽 상단의 톱니바퀴 아이콘을 클릭한 후 “Code Editor (Ace)” 를 선택하여 “Soft Tabs” 설정을 변경하면 된다. (이 때 “저장” 버튼을 클릭하지 않아도 바로 적용된다는 점에 주목하라.)

images/figures/cloud9_new_workspace
그림 1.2: Cloud9 에서 새 워크스페이스 만들기.
images/figures/cloud9_two_spaces
그림 1.3: Cloud9 에 들여쓰기를 두개의 공백 문자로 설정하기.

1.2.2 레일스 설치하기

섹션 1.2.1 의 개발 환경은 레일스를 제외한 모든 소프트웨어를 갖춘 상태다. 8 레일스를 설치하려면 RubyGems 패키지 관리자에서 제공하는 gem 명령어를 사용해야 한다. 예제 1.1 의 명령어를 커맨드라인 터미널에 입력하면 된다. (독자의 로컬 머신에서 개발한다면 터미널 창에 명령을 입력하고, 클라우드 통합 개발환경를 사용한다면 그림 1.1 에 보이는 커맨드라인 영역에 입력하면 된다.)

목록 1.1: 특정 버전의 레일스 설치하기
```
$ gem install rails -v 4.2.0
```

-v 옵션으로 특정 버전의 레일스를 설치하도록 지정할 수 있다. 다른 버전을 설치할 경우 이 튜토리얼에 나오는 결과와 다른 출력이 나올 수 있다.

1.3 첫번째 애플리케이션

컴퓨터 프로그래밍의 오랜 전통에 따라 첫번째 애플리케이션의 목표는 "hello, world" 프로그램을 작성하는 것이다. 개발 환경(섹션 1.3.4)과 실제 인터넷(섹션 1.5) 웹 페이지에 "hello, world!" 를 출력하는 프로그램을 만들 것이다.

거의 대부분의 경우 레일스 애플리케이션 개발은 rails new 명령을 실행하는 것으로 시작한다. 이 간단한 명령은 여러분이 선택한 디렉토리에 레일스 애플리케이션의 기본 골격을 생성한다. 시작하기 전에 섹션 1.2.1 에서 추천한 Cloud9 통합 개발환경를 사용하지 않을 경우에는, 레일스 프로젝트가 위치할 workspace 디렉토리가 없으며 새로 만든 후(목록 1.2) 그 디렉토리로 이동해야 한다. (목록 1.2 에서 유닉스 명령어 cd 와 mkdir 를 사용하였다. 이 명령어가 익숙하지 않을 경우 글상자 1.3 을 읽어라.)

목록 1.2: 레일스 프로젝트가 위치할 workspace 디렉토리 생성(클라우드 사용시 불필요함).

```
$ cd                # 홈 디렉토리로 이동.
$ mkdir workspace   # 디렉토리 생성.
$ cd workspace/     # workspace 디렉토리로 이동.
```

글상자 1.3. 유닉스 커맨드라인 특강

윈도우를 주로 사용해 온 독자나 (소수이지만 빠르게 늘어나고 있는) 매킨토시 OS X 을 사용하는 독자들에게 유닉스 커맨드라인은 생소할 수 있다. 추천한 대로 클라우드 환경에서 개발한다면 바로 Bash 라 부르는 표준 쉘 커맨드라인 인터페이스를 통해 유닉스(리눅스) 커맨드라인을 사용할 수 있다.

커맨드라인의 기본 개념은 간단하다. 사용자는 짧은 명령어를 실행하여 디렉토리 생성(mkdir), 파일 복사 및 이동 (mv 와 cp), 디렉토리 이동(cd)같은 여러 작업을 수행하는 것이다. 그래픽 사용자 인터페이스(GUI)에 익숙한 대부분의 사람들에게 커맨드라인은 원시적으로 보일 수도 있겠지만, 보기와는 달리, 커맨드라인은 개발자의 도구 중 가장 강력한 도구의 하나라 할 수 있다. 사실 숙련된 개발자의 화면에서는 커맨드라인 쉘을 실행하는 터미널 창이 항상 열려 있다.

이 튜토리얼에서는 표 1.1 로 요약한 가장 많이 쓰이는 몇 가지 유닉스 커맨드라인 명령만 사용한다. 유닉스 커맨드라인에 대해 좀 더 자세히 알고자 하면 마크

베이츠(Mark Bates)의 커맨드라인 완전 정복 (무료 온라인 버전이나 유료 전자책/스크린캐스트로 볼 수 있다.).

기능설명	명령어	예
파일 목록	ls	$ ls -l
디렉토리 생성	mkdir 〈dirname〉	$ mkdir workspace
하위 디렉토리로 이동	cd 〈dirname〉	$ cd workspace/
상위 디렉토리로 이동		$ cd ..
홈 디렉토리로 이동		$ cd ~ or just $ cd
홈 디렉토리를 포함한 경로로 이동		$ cd ~/workspace/
파일 이동 (파일 이름 변경)	mv 〈source〉 〈target〉	$ mv README.rdoc README.md
파일 복사	cp 〈source〉 〈target〉	$ cp README.rdoc README.md
파일 삭제	rm 〈file〉	$ rm README.rdoc
빈 디렉토리 삭제	rmdir 〈directory〉	$ rmdir workspace/
비어있지 않은 디렉토리 삭제	rm -rf 〈directory〉	$ rm -rf tmp/
연속적으로 & 파일 내용 출력	cat 〈file〉	$ cat ~/.ssh/id_rsa.pub

표 1.1: 기초 유닉스 명령어.

다음 단계로 로컬 시스템이나 클라우드 통합 개발환경 어느 쪽에서 작업하더라도, 목록 1.3 에 나온 명령으로 첫번째 애플리케이션을 생성한다. 목록 1.3 에서 레일스 버전 넘버(_4.2.0_)를 명령의 일부로 입력한 것에 주의하라. 이것은 목록 1.1 에서 설치한 레일스와 동일한 버전을 설치했다는 것을 확인해 준다. (만약 목록 1.3 에서 “Could not find ’railties”’와 같은 에러를 출력한다면, 다른 버전의 레일스가 설치되었다는 의미다. 이 경우 목록 1.1 에 나온 명령 대로 입력했는지 다시 한번 확인하라.)

목록 1.3: rails new 실행 (특정 버전 넘버 지정).

```
$ cd ~/workspace
$ rails _4.2.0_ new hello_app
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/javascripts/application.js
      create  app/assets/stylesheets/application.css
      create  app/controllers/application_controller.rb
      .
      .
      .
      create  test/test_helper.rb
      create  tmp/cache
      create  tmp/cache/assets
      create  vendor/assets/javascripts
      create  vendor/assets/javascripts/.keep
      create  vendor/assets/stylesheets
      create  vendor/assets/stylesheets/.keep
         run  bundle install
Fetching gem metadata from https://rubygems.org/.........
Fetching additional metadata from https://rubygems.org/..
Resolving dependencies...
Using rake 10.3.2
Using i18n 0.6.11
.
.
.
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is installed.
         run  bundle exec spring binstub --all
* bin/rake: spring inserted
* bin/rails: spring inserted
```

목록 1.3 에서 보듯이, rails new 명령을 실행하면 파일들을 생성한 후 자동으로 bundle install 명령을 수행한다. 이게 무엇을 의미하는지 섹션 1.3.1 에서 자세히 다룰 것이다.

rails 명령이 여러 파일들과 디렉토리를 생성하는 것을 볼 수 있다. 여기서 생성된 표준 디렉토리, 파일 구조는 (그림 1.4) 레일스가 가진 장점 중 하나다. 이로 인해 제로 상태에서 동작하는 애플리케이션이 생성된다. 더욱이 이러한 구조는 모든 레일스 애플리케이션에서 동일하므로 다른 사람이 작성한 코드를 보더라도 쉽게 알아 볼 수 있다. 표 1.2 에서 레일스 기본 파일에 대해 요약하였고, 이 책의 나머지 부분을 진행하면서 이 파일과 디렉토리에 대해 좀 더 학습하게 될 것이다. 특히 섹션 5.2.1 에서는 캐스케이딩 스타일 시트(CSS)나 자바스크립트 파일들의 구성을 쉽게 해주는 애셋 파이프라인에 대해 알아보며 app/assets 디렉토리에 대해 살펴 볼 것이다.

images/figures/directory_structure_rails_3rd_edition

그림 1.4: 새로 생성된 레일스 앱의 디렉토리 구조

파일/디렉토리	용도
app/	애플리케이션 핵심 코드. 모델, 뷰, 컨트롤러, 헬퍼를 포함한다.
app/assets	애플리케이션에 포함된 애셋(캐스케이딩 스타일시트(CSS), 자바스크립트, 이미지 파일)
bin/	이진 실행 파일들
config/	애플리케이션 설정 파일
db/	데이터베이스 관련 파일
doc/	애플리케이션 관련 문서
lib/	라이브러리 모듈
lib/assets	라이브러리에 포함된 애셋(캐스케이딩 스타일시트(CSS), 자바스크립트, 이미지 파일)
log/	애플리케이션 로그 파일
public/	(웹 브라우저로 접근 가능한) 공개 디렉토리, 에러 페이지 등이 위치함.
bin/rails	코드 생성, 콘솔 세션 접속, 서버 기동 등을 수행하는 프로그램
test/	애플리케이션 테스트 코드
tmp/	임시 파일들
vendor/	플러그인이나 젬 등의 서드파티 코드

vendor/assets	캐스케이딩 스타일 시트(CSS)와 자바스크립트 그리고 이미지 등의 서드 파티 애셋 파일들
README.rdoc	애플리케이션에 대한 간단한 소개
Rakefile	rake 명령어를 통한 유틸리티 태스크 정의
Gemfile	애플리케이션에서 사용하는 젬 라이브러리 의존성 정의
Gemfile.lock	앱의 복사본이 모두 동일 버전의 젬을 사용하도록 작성된 젬 목록 파일
config.ru	Rack 미들웨어용 설정 파일
.gitignore	Git 에서 무시해야 할 파일 패턴을 정의한 파일

표 1.2: 기본 레일스 디렉토리 구조에 대한 요약.

1.3.1 Bundler

레일스 애플리케이션을 생성한 후 다음 절차는 애플리케이션에서 사용하는 젬을 Bundler 로 설치하는 것이다. 섹션 1.3 에서 언급했듯이, Bundler 는 (bundle install 명령을 통해) rails 명령과 함께 자동으로 실행된다. 이번 섹션에선 애플리케이션에서 사용하는 젬 목록을 수정 한 후 다시 한번 Bundler 를 실행할 것이다. Gemfile 파일을 텍스트 편집 프로그램으로 열어라. (클라우드 통합 개발환경을 사용할 때엔, 파일 네비게이터에서 화살표를 클릭하여 sample 애플리케이션을 열고 Gemfile 아이콘을 더블클릭한다.) 버전 넘버와 상세 내용이 조금 다를 수 있지만, 파일 내용은 그림 1.5 과 목록 1.4 과 같을 것이다. (이 파일의 코드는 루비 언어로 되어 있는데, 이 시점에서 문법을 모른다고 걱정할 필요는 없다. 4 장에서 좀 더 깊게 다룰 것이다.) 혹시 파일과 디렉토리가 그림 1.5 처럼 보이지 않는다면, 파일 네비게이터의 톱니바퀴 아이콘을 클릭한 후 보이는 메뉴에서 “Refresh File Tree” 항목을 선택하라. (이후에도 파일이나 디렉토리가 예상과 달리 보이지 않는 경우 새로 고침을 한다.)

images/figures/cloud9_gemfile

그림 1.5: Gemfile 을 텍스트 에디터로 연 모습.

목록 1.4: hello_app 디렉토리 상의 기본 Gemfile .

```
source 'https://rubygems.org'


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.0'
```

```
# Use sqlite3 as the database for Active Record
gem 'sqlite3'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0.1'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .js.coffee assets and views
gem 'coffee-rails', '~> 4.0.0'
# See https://github.com/sstephenson/execjs#readme for more supported runtime
s
# gem 'therubyracer', platforms: :ruby

# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read more:
# https://github.com/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc

# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use Unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development

group :development, :test do
  # Call 'debugger' anywhere in the code to stop execution and get a
  # debugger console
  gem 'byebug'
```

SungHo Choi 6/12/2016 6:00 PM

Comment [1]: Gem 자체의 버그로, 원본 tutorial 과는 다르지만 5.0.2+ 버전으로 적는 것으 좋을 것 같습니다.
http://stackoverflow.com/questions/29716284/uninitialized-constant-sassscript-nameerror

```
  # Access an IRB console on exceptions page and /console in development
  gem 'web-console', '~> 2.0.0.beta2'

  # Spring speeds up development by keeping your application running in the
  # background. Read more: https://github.com/rails/spring
  gem 'spring'
end
```

많은 코드라인이 해시 심벌 #로 주석 처리되어 있다. 주서 처리된 내용은 흔히 사용하는 젬에 대한 Bundler 의 작성 예다. 지금은 기본 젬 외에 다른 것을 추가 할 필요가 없다.

다음의 코드에서 처럼 gem 명령에 버전 번호를 따로 지정하지 않는 한, Bundler 는 해당 젬의 최신 버전을 설치한다.

```
gem 'sqlite3'
```

레일스가 사용할 젬 버전 범위를 강제로 지정하는 데에는 두 가지 방법이 있다. 첫 번째 방법은 다음과 같다.

```
gem 'uglifier', '>= 1.3.0'
```

이 설정은 1.3.0 버전 보다 같거나 가장 최신 버전의 uglifier 젬 (애셋 파이프라인에서 파일들을 압축한다.)을 설치하도록 한다. 7.2 버전 예에서도 마찬가지다. 두번째 방법은 다음과 같다.

```
gem 'coffee-rails', '~> 4.0.0'
```

이렇게 정의하면 coffee-rails 젬을 설치할 때 4.0.0 보다 최신이지만 4.1 보다는 낮은 버전을 설치한다. 다시 말해서, >=로 표기하면 무조건 최신 버전의 gem 을 설치하고, ~> 4.0.0 로 표기하면 지정된 버전의 마이너 업데이트 버전만 설치하고(4.0.0 버전에서 4.0.1 버전으로), 메이저 업데이트 버전(4.0 버전에서 4.1 버전으로)으로의 업데이트는 수행하지 않는다.. 경험상 마이너 버전 업그레이드도 이상 동작을 일으킬 수 있다. 루비온레일스 튜토리얼에서는 부작용을 피하기 위해 모든 젬 버전을 지정해 사용하고 있다. Gemfile 에서 ~> 표기법을 사용하면 어떤 젬 라이브러리라도 최신 버전을 사용 할 수 있다(초보 수준을 넘어선 개발자에게 추천한다.). 하지만 이 경우 튜토리얼을 진행하면서 다른 결과가 나올 수 있다는 점을 밝혀둔다.

목록 1.5 처럼 결과가 나오도록 Gemfile 내용을 목록 1.4 처럼 변경한다.
허로쿠(섹션 1.5)에서 데이터베이스를 설정할 때 발생할 수 있는 충돌을 막기 위해

sqlite3 젬을 개발 환경과 테스트 환경(섹션 7.1.1)에서만 사용되도록 한 것에 유의하라.

목록 1.5: 각 루비 젬 파일이 특정 버전을 사용하도록 강제한 Gemfile

```
source 'https://rubygems.org'

gem 'rails',                '4.2.0'
gem 'sass-rails',           '5.0.1'
gem 'uglifier',             '2.5.3'
gem 'coffee-rails',         '4.1.0'
gem 'jquery-rails',         '4.0.3'
gem 'turbolinks',           '2.3.0'
gem 'jbuilder',             '2.2.3'
gem 'sdoc',                 '0.4.0', group: :doc

group :development, :test do
  gem 'sqlite3',     '1.3.9'
  gem 'byebug',      '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',      '1.1.3'
end
```

목록 1.5 와 같이 Gemfile 파일을 수정한 후, bundle install 명령으로 설치한다:9

```
$ cd hello_app/
$ bundle install
Fetching source index for https://rubygems.org/
.
.
.
```

bundle install 명령이 실행되는 데 몇 분 정도 시간이 소요될 것이다. 실행이 완료되면 서버를 실행 할 준비가 끝난 것이다.

1.3.2 rails server

섹션 1.3 에서 rails new 명령을, 섹션 1.3.1 에서 bundle install 명령을 실행하는 것으로 애플리케이션을 실행할 수 있게 되었다. 어떻게 이것이 가능한 것일까?

레일스는 애플리케이션 개발에 편리하도록 로컬 웹 서버 프로그램(혹은 script)를 포함하고 있다. 정확한 명령은 시스템마다 다를 수 있는데, 로컬 개발 환경에선 rails server (목록 1.6), Cloud9 에서는 추가로 IP 바인딩 주소와 포트 번호 를 지정해 주어 외부에서 접근할 수 있는 주소를 레일스 서버로 알려 준다(목록 1.7).10 (Cloud9 는 전용 환경 변수 $IP 와 $PORT 를 사용해서 동적으로 IP 주소와 포트 번호를 할당한다. 이들의 정확한 값을 확인하려면 echo $IP 나 echo $PORT 명령을 커맨드라인에서 입력해 본다.)

목록 1.6: 로컬 머신에서 레일스 서버 실행하기

```
$ cd ~/workspace/hello_app/
$ rails server
=> Booting WEBrick
=> Rails application starting on http://localhost:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

목록 1.7: 클라우드 통합 개발환경에서 레일스 서버 실행하기

```
$ cd ~/workspace/hello_app/
$ rails server -b $IP -p $PORT
=> Booting WEBrick
=> Rails application starting on http://0.0.0.0:8080
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
```

둘 중 어느 쪽이든, 그림 1.6 과 그림 1.7 과 같이 두번째 터미널 탭에서 rails server 명령을 실행하기 바란다. (이미 첫번째 탭에서 실행했다면, Ctrl-C 를 눌러 서버를 종료한다.)11 로컬 서버에서는 웹브라우저의 페이지 주소를 http://localhost:3000/ 로 입력하고, 클라우드 통합 개발환경을 사용하는 경우, Share 메뉴에서 Application 주소를 클릭해서 페이지를 열면 된다(그림 1.8). 어느 쪽이든 결과 화면은 그림 1.9 과 같이 나올 것이다.

images/figures/new_terminal_tab

그림 1.6: 새 터미널 탭에서 열기.

images/figures/rails_server_new_tab

그림 1.7: 레일스 서버를 별도의 탭에서 실행하기.

images/figures/share_workspace

그림 1.8: 클라우드 워크스페이스에서 레일스 서버 실행하기.

SungHo Choi 6/12/2016 6:07 PM

Comment [2]: 이 부분에서 rails server 실행이야 당연히 'rails server' command 이다.. 라고 생각하고 넘어갔는데 놓치면은 아래 예제들을 쫓아가며 c9 에서 server 를 실행하는데 어려움이 많습니다. 더 강조해서 독자가 놓치지 않도록 하는 편이 좋을 것 같습니다.

SungHo Choi 6/12/2016 6:00 PM

Comment [3]: 내용이 '클라우드 워크스페이스에서 레일스 서버 시작하기'와 무관한 Share Workspace 인데다가, 어떻게 하는지에 대한 설명이 미흡합니다.

images/figures/riding_rails_3rd_edition
그림 1.9: rails server 명령으로 제공되는 레일스 기본 페이지.

첫번째 애플리케이션의 상세 정보를 보려면 “About your application’s environment” 링크를 클릭한다. 각 버전 정보는 다를 수 있겠으나 결과는 그림 1.10 과 같을 것이다. 이후로 레일스 기본 페이지는 사용하지 않겠지만, 지금은 일단 제대로 돌아간다는 것을 확인하는 용도로는 충분하다. 기본 화면은 섹션 1.3.4 에서 새로 만든 페이지로 대체할 것이다.

images/figures/riding_rails_environment_3rd_edition
그림 1.10: 애플리케이션 개발환경 정보를 포함하는 기본 화면.

1.3.3 모델-뷰-컨트롤러 (MVC)

개발 초기 단계에서도 레일스 애플리케이션이 어떻게 동작하는지 큰 그림을 이해하는 것이 도움이 된다. (그림 1.11). 레일스 애플리케이션의 표준 구조를 보면 (그림 1.4) 애플리케이션 디렉토리 app/ 가 있고 그 안에 다음 세 개의 하위 디렉토리가 있다. models, views, controllers. 이것은 레일스가 모델-뷰-컨트롤러 (MVC) 구조 패턴을 따르고 있다는 걸 보여준다. 이 구조 패턴은 “도메인 로직” (“비즈니스 로직”으로도 불리는)을 그래픽 사용자 인터페이스(GUI) 상의 입력/출력으로부터 강제로 분리해 준다. 웹 애플리케이션에서 “도메인 로직” 은 사용자, 작성글, 제품 같은 데이터 모델들을 의미하고, GUI 는 웹 브라우저로 표현되는 웹 페이지를 의미한다.

레일스 애플리케이션을 사용할 때, 브라우저는 요청(request)을 보내고 웹 서버는 이를 받아 레일스 컨트롤러에 전달하여 요청을 처리한다. (어떤 경우에) 컨트롤러는 이 요청에 대해 즉각적으로 뷰를 렌더링하는데, 이 뷰는 템플릿으로써 HTML 로 변환되어 브라우저에 응답으로 보내진다. 동적 사이트에서는 더 흔하게 컨트롤러가 모델과 상호 작용하는데, 이것은 루비 객체로써 (사용자 같은) 사이트의 구성 요소를 나타내고 데이터베이스와의 통신을 담당한다 . 모델을 호출 한 후 컨트롤러는 뷰를 렌더링하여 브라우저에 보여질 웹 페이지 전체를 HTML 로 반환한다.

mvc_schematic
그림 1.11: 모델-뷰-컨트롤러 (MVC) 구조의 도해.

이 주제가 추상적으로 보일 수 있는데, 이번 섹션에서 계속 설명할 것이므로 걱정하지 않아도 된다. 섹션 1.3.4 은 임시로 만드는 첫번째 MVC 애플리케이션이다. 섹션 2.2.2 에서는 toy 앱의 관점에서 MVC 를 깊게 다룰 것이다. 마지막으로 sample 앱에서는 MVC 의 모든 관점을 사용하여, 섹션 3.2 에서는 컨트롤러와 뷰를, 섹션 6.1 에서는 모델을, 섹션 7.1.2 에서는 이들이 함께 동작하는 것을 다룰 것이다.

1.3.4 Hello, world!

MVC 프레임워크로 첫번째 애플리케이션을 미세하게(wafer-thin) 변경하여 "hello, world!"를 출력하는 컨트롤러 액션을 추가 할 것이다. (섹션 2.2.2 에서 컨트롤러 액션에 대해 좀더 살펴 볼 것이다.) 이 섹션의 목적은 그림 1.9 의 레일스 기본 페이지를 "hello, world" 페이지로 바꾸는 것이다.

SungHo Choi 6/12/2016 5:59 PM
Comment [4]: https://en.wikipedia.org/wiki/Mr_Creosote 로의 link 가 왜 있는건지 모르겠습니다..

컨트롤러 액션은 이름 자체에서 추측할 수 있듯이 컨트롤러 내부에 정의 된다. 이 액션을 hello 로 부르기로 하고 Application 컨트롤러에 정의하겠다. 현재로선 Application 컨트롤러가 유일한 컨트롤러 클래스다. 다음의 명령을

```
$ ls app/controllers/*_controller.rb
```

실행하면 현재 작성된 컨트롤러를 확인할 수 있다. (2 장 부터는 컨트롤러를 추가할 것이다.) 목록 1.8 은 "hello, world!"를 출력하는 render 함수를 사용하는 hello 액션을 보여 준다. (루비 문법에 대해서는 걱정하지 않길 바란다. 4 장에서 다룰 예정이다.)

목록 1.8: Application 컨트롤러에 hello 액션 추가 하기.

SungHo Choi 6/12/2016 6:11 PM
Comment [5]: 상위 본 책에 대한 설명에서, '중요한 부분, 이전 예제에서 달라진 부분' 에 대한 highlight 를 설명한 부분이 있는데요, def hello ... end 에 highlight 가 빠져있습니다.

app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # 예외를 발생시켜 CSRF 공격을 방지한다.
  # API 로 사용할 경우에는, :null_session 을 대신 사용할 수 있다.
  protect_from_forgery with: :exception

  def hello
    render text: "hello, world!"
  end
end
```

지정한 문자열을 출력하는 액션을 추가했으니, 이제 레일스가 그림 1.10 에서 보는 기본 페이지 대신에 방금 추가한 액션을 사용하도록 알려 줘야 한다. 그러기 위해 그림 1.11 에서 보는 바와 같이 브라우저의 요청을 어느 컨트롤러에서 처리할 것인지 결정하는 레일스 라우터를 컨트롤러를 실행하기 전에 먼저 수정해야 한다. (그림 1.11 에서 라우터를 편의상 생략했는데, 이 부분은 섹션 2.2.2 에서 좀 더 다루겠다.) 여기서는 루트 라우트를 변경하여 루트 URL 을 설정한다. URL 은 http://www.example.com/ 과 같은 주소(마지막 슬래시 다음 단어 뒤에 아무것도 없을 때까지)를 의미하는데, 루트 URL 은 이를 짧게 줄여 / ("슬래시")로 표시한다.

목록 1.9 과 같이 레일스 라우트 파일(config/routes.rb)은 루트 경로가 주석처리 되어 있다. 여기서 “welcome” 은 컨트롤러 이름이고 “index” 는 컨트롤러의 액션을 의미한다. 루트 경로를 활성화하려면, 해시 문자를 삭제해서 주석을 해제하고 목록 1.10 처럼 코드를 변경하여 Application 컨트롤러 hello 액션을 사용하도록 해야 한다. (섹션 1.1.2 에서 언급했지만, 수직으로 배열된 점은 코드를 생략한 부분이니 그대로 복사하면 안된다.)

목록 1.9: 기본 루트 라우트 (주석 처리한 부분). config/routes.rb

```
Rails.application.routes.draw do
  .
  .
  .
  # You can have the root of your site routed with "root"
  # root 'welcome#index'
  .
  .
  .
end
```

목록 1.10: 루트 경로 설정하기. config/routes.rb

```
Rails.application.routes.draw do
  .
  .
  .
  # You can have the root of your site routed with "root"
  root 'application#hello'
  .
  .
  .
end
```

목록 1.8 과 목록 1.10 과 같이 코드를 작성하면, 루트 라우트를 요청할 때 “hello, world!”를 반환한다(그림 1.12).

images/figures/hello_world_hello_app
그림 1.12: 브라우저에서 “hello, world!” 보기.

1.4 Git 버전 관리

지금까지는 레일스 애플리케이션을 생성한 후 실제로 동작하도록 만들었다. 여기서 잠시 시간을 내어, 기술적인 선택 사항이지만, 숙련된 개발자들 사이에선 필수 기술로 여기는 버전 관리 시스템으로 코드를 관리하도록 하겠다. 버전 관리 시스템은 프로젝트 코드의 변경을 추적할 수 있게 해주고, 협업을 쉽게해 주고, 의도하지 않은 오류(실수로 파일을 지운다던가)에서 복원 도 가능하게 해준다. 버전 관리 시스템을 사용할 줄 아는 것은 프로페셔널 소프트웨어 개발자라면 필수 역량이다.

어떤 종류의 버전 관리 시스템을 사용할 것인가에 대해서는 여러 가지 옵션이 있겠지만, 거의 모든 레일스 커뮤니티는 리누스 토발즈가 리눅스 커널을 개발하기 위해 개발한 Git 을 표준으로 사용하고 있다. Git 은 이 책에서 다루기엔 너무 큰 주제이기에 여기에선 겉핥기 정도로 다룰 것이다. Git 을 다루는 수많은 무료 자료들 중 스콧 샤콘의 Pro Git 을 추천한다. 레일스 세상에서 널리 쓰이기 뿐만 아니라, 코드를 쉽게 백업하고 공유할 수 있으며(섹션 1.4.3) 당장 첫 장에서 애플리케이션을 배포하는데 사용하므로(섹션 1.5) 각자의 소스를 Git 으로 버전 관리하는 것을 강력하게 추천한다.

SungHo Choi 6/12/2016 6:08 PM
Comment [6]: Pro Git 의 한글 번역판이 존재하는 걸로 알고 있습니다.함께 적어주면 좋을 것 같습니다.

1.4.1 설치 및 설정

섹션 1.2.1 에서 추천한 클라우드 통합 개발환경는 Git 을 기본으로 포함하고 있다. 따라서 이 경우에는 아무것도 설치 할 필요가 없다. 그 외에는 InstallRails.com(섹션 1.2)에 나온 방법을 따라 각자의 시스템에 Git 을 설치한다.

최초 시스템 설정

Git 을 사용하기 전, 처음 한번은 설정 단계를 거쳐야 한다. 시스템 설정은 이름 그대로 사용하는 컴퓨터마다 해줘야 한다.

```
$ git config --global user.name "Your Name"
$ git config --global user.email your.email@example.com
$ git config --global push.default matching
$ git config --global alias.co checkout
```

위의 Git 설정에 사용된 이름과 이메일 주소는 모든 소스 저장소에서 공통적으로 적용될 것이다. (맨 첫 두 줄은 필수로 입력해야 한다. 세번째 줄은 Git 의 장차 나올 새 버전에 대한 호환성을 위해 추가 되었다. 네번째 줄은 checkout 명령어를 co 로 짧게 줄여준다. 시스템간의 호환성을 위해서는 co 설정을 할 필요는 없으며, 이

튜토리얼에서는 전체 명령어 checkout 을 사용하겠지만 저자는 실제 개발에서 git co 을 사용한다.)

최초 저장소 설정

새 소스 저장소(짧게 줄여 저장소라 하겠다.)를 추가할 때마다 해줘야 할 설정이 있다. 우선 애플리케이션의 루트 디렉토리로 이동하여 저장소를 초기화한다.

```
$ git init
Initialized empty Git repository in /home/ubuntu/workspace/hello_app/.git/
```

그 다음엔 git add -A 명령으로 프로젝트 파일을 저장소에 등록한다.

```
$ git add -A
```

SungHo Choi 6/12/2016 6:12 PM
Comment [7]: Command option 을 축약어로 적는 것보다는 초심자들이 이해하기 수월하도록 늘어 적는 것이 어떨까요? (git add –all 과 같이..) 다른 command 들에도 적용을 바라는 의견입니다.

이 명령은 .gitignore 파일에 정의된 패턴과 일치하는 파일/디렉토리를 제외한 모든 파일을 Git 에 등록시켜준다. rails new 명령이 실행될 때 자동으로 .gitignore 파일이 생성 되지만, 필요에 따라 패턴을 추가할 수 도 있다. 12

파일이 추가 되면 이 파일이 처음에는 프로젝트의 변경점을 저장하는 스테이징 영역에 들어가게 된다. 스테이징 영역에 들어간 파일들을 확인하려면 status 명령을 사용한다.

```
$ git status
On branch master

Initial commit

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

  new file:   .gitignore
  new file:   Gemfile
  new file:   Gemfile.lock
  new file:   README.rdoc
  new file:   Rakefile
  .
  .
  .
```

(결과 내용이 길어서 수직으로 배열된 점으로 출력이 생략됨을 나타냈다)

Git 에 변경내용을 저장하려면 commit 명령을 사용한다.

```
$ git commit -m "Initialize repository"
[master (root-commit) df0a62f] Initialize repository
.
.
.
```

-m 플래그는 커밋 시에 메시지를 추가해 준다. -m 플래그를 생략할 경우 Git 은 메시지를 입력 하도록 시스템의 기본 편집기 프로그램을 실행한다. (이 책의 모든 예제에서 -m 플래그를 사용한다.)

여기서 주목할 점은 Git 커밋은 로컬에서만 이뤄진다는 점이다. 기록은 커밋이 실행된 머신에서만 수행된다. 원격 저장소에 변경 사항을 푸시(git push 명령어를 사용한다)하는 것은 섹션 1.4.4 에 서 살펴 볼 것이다.

커밋 메시지 목록을 보려면 log 명령을 사용한다.

```
$ git log
commit df0a62f3f091e53ffa799309b3e32c27b0b38eb4
Author: Michael Hartl
Date:   Wed August 20 19:44:43 2014 +0000

    Initialize repository
```

여러분의 로그 히스토리 길이에 따라 q 를 눌러야 로그 보기를 종료할 수도 있다.

1.4.2 Git 의 어떤 점이 좋은가?

버전 관리 시스템을 사용해 본 적이 없다면, 왜 좋은 지 모를 것이다. 예를 들어 설명해 보겠다. 실수로 프로젝트가 변경된 경우를 상상해 보자, 예를 들어 핵심 디렉토리 중 하나인 app/controllers/를 삭제해 보겠다.

```
$ ls app/controllers/
application_controller.rb  concerns/
$ rm -rf app/controllers/
$ ls app/controllers/
ls: app/controllers/: No such file or directory
```

app/controllers/ 디렉토리 내용을 보여주는 유닉스 명령어 ls 와 이 파일을 삭제하는 rm 명령을 실행하였다. (표 1.1). -rf 플래그는 "반복적이고 강제로"란 의미로 삭제 여부를 사용자에게 일일이 묻지 않고 삭제 대상의 파일, 디렉토리, 하위 디렉토리 전체를 삭제하라는 의미다.

status 옵션으로 변경 사항을 확인한다.

```
$ git status
On branch master
Changed but not updated:
  (use "git add/rm ..." to update what will be committed)
  (use "git checkout -- ..." to discard changes in working directory)

      deleted:    app/controllers/application_controller.rb

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

파일이 삭제된 것을 볼 수 있는데, 삭제는 사실 "작업 트리"에서만 일어난 것이다. 커밋을 아직 하지 않은 상태라는 의미다. 이렇게 되면 checkout 명령과 강제로 변경 사항을 덮어쓰라는 의미의 -f 플래그를 사용하여 파일 삭제를 되돌릴 수 있다는 의미이다.:

```
$ git checkout -f
$ git status
# On branch master
nothing to commit (working directory clean)
$ ls app/controllers/
application_controller.rb  concerns/
```

삭제 되었던 파일과 디렉토리가 복구 되었다. 이제 안심이다!

1.4.3 Bitbucket

Git 을 사용해서 프로젝트를 버전 관리하고 있으므로, 이제 이 코드들을 Git 저장소를 호스팅하고 공유하는데 최적화된 Bitbucket 서비스로 푸시하도록 하자. (이 튜토리얼의 이전 판에서는 GitHub 을 사용했었다. 글상자 1.4 에서 변경 이유에 대해서 설명해 놓았다.) Bitbucket 서비스에 Git 소스 저장소의 복제본을 올리는 데에는 두 가지 목적이 있다. 첫째는 전체 코드와 커밋 내역 전체에 대한 백업을 하는 것이고, 둘째는 향후 다른 동료와 협업을 쉽게 하기 위해서다.

글상자 1.4. GitHub 와 Bitbucket

Git 저장소 호스팅 사이트 중 가장 인기 있는 곳은 GitHub 과 Bitbucket 이다. 두 서비스는 서로 유사한 부분이 많은데, 두 사이트 모두 동료와 협업도 가능하며 코드를 검색하거나 둘러보기 편하게 기능을 제공하는 Git 저장소 호스팅 서비스다. 가장 큰 차이(이 튜토리얼에서)는 GitHub 의 오픈 소스 저장소용 무료 계정은 무제한의 저장소와 협업자를 설정할 수 있지만, 비공개 저장소를 사용하려면 사용 요금을 내야 하는데, Bitbucket 은 무료 계정에서도 비공개 저장소를 제공한다는 것이다. 어느 서비스를 소스 저장소로 사용할지는 각자의 상황에 따라 결정하면 된다.

이 책의 이전 판에서는 오픈소스를 지지하는 의미로 GitHub 을 사용했지만, 보안의 중요성을 고려해서 모든 웹 애플리케이션 소스 저장소는 비공개여야 한다고 생각하게 되었다. 웹 애플리케이션 저장소에 암호키나 비밀번호 같은 민감한 정보가 포함될 수 있고 악의적인 사용자들에게 노출 될 수 있기 때문이다. 물론 이런 정보들을 안전하게 다룰 수(해당 파일을 Git 에서 무시하게 하는 등의 방법으로) 있겠지만 이런 것은 많은 경험을 필요로 한다.

이 튜토리얼을 통해 만들게 될 sample 애플리케이션이 웹에 공개 된다 해도 딱히 문제가 발생하지 않겠지만, 이런 접근은 안이한 것이다. 최대한 보안을 고려하여 모든 저장소는 비공개되야 한다는 것을 강조하는 바이다. 앞에서 언급했지만 GitHub 는 비공개 저장소 사용에 비용을 부과하는데 반해 Bitbucket 은 무료로 제공한다. 따라서 Github 보다 Bitbucket 이 우리 목적에 맞다고 판단했다.

Bitbucket 을 사용하는 건 단순하다.

1. Bitbucket 계정이 없다면 생성한다.
2. 공개 키를 클립보드에 복사한다. 목록 1.11 과 같이, 클라우드 통합 개발환경 사용자는 cat 명령어로 공개 키를 확인 할 수 있다. 자신의 시스템에서 작업하는 경우 목록 1.11 의 명령을 입력해도 아무런 결과가 없을 것이다. 이런 경우 Bitbucket 계정에 공개 키 설치하기 문서를 따라한다.
3. Bitbucket 페이지의 오른쪽 상단의 아바타 이미지를 클릭하고 “Manage account” 를 선택한 뒤 “SSH keys” 항목을 클릭한다(그림 1.13).

목록 1.11: cat 명령으로 공개 키 출력하기.

```
$ cat ~/.ssh/id_rsa.pub
```

images/figures/add_public_key

그림 1.13: SSH 공개 키 추가 하기.

공개 키를 추가 한 후, 그림 1.14 처럼 “Create” 버튼을 클릭하여 create a new repository 페이지로 이동한다. 프로젝트 정보를 입력할때 반드시 “This is a private repository.” 항목을 체크 한다. “Create repository” 버튼을 클릭한 후, “Command line 〉 I have an existing project” 링크를 클릭하여 지시한 대로 작업을 진행한다(이것은 목록 1.12 와 같이 보일 것이다). (목록 1.12 와 다른 결과라면 공개 키가 제대로 추가 되지 않은 것이다. 앞의 과정을 다시 해보자.) 저장소로 코드를 푸시 할 때 “Are you sure you want to continue connecting (yes/no)?” 질문에 yes 를 선택하자.

images/figures/create_first_repository_bitbucket
그림 1.14: Bitbucket 으로 첫번째 애플리케이션 저장소 생성하기.

목록 1.12: Bitbucket 을 리모드 저장소로 추가하고 소스 푸시 하기.

```
$ git remote add origin git@bitbucket.org:〈username〉/hello_app.git
$ git push -u origin --all # pushes up the repo and its refs for the first time
```

목록 1.12 의 명령은 Git 이 Bitbucket 을 origin 저장소로 지정하도록 한다. 그 다음은 작업 중인 로컬 저장소를 원격 origin 저장소로 푸시하란 의미다. (-u 플래그가 어떤 의미인지 신경 쓰지 않아도 되지만, 궁금하다면 “git set upstream” 으로 검색해보면 된다.) 물론 〈username〉 은 실제 계정 이름으로 변경해야 한다. 저자의 경우를 예를 들면 다음과 같다.

```
$ git remote add origin git@bitbucket.org:mhartl/hello_app.git
```

이렇게 하면 Bitbucket 에 hello_app 저장소 페이지가 생성 되고 파일 탐색, 전체 커밋 내역 등을 볼 수 있다(그림 1.15).

images/figures/bitbucket_repository_page
그림 1.15: Bitbucket 저장소 페이지.

1.4.4 브랜치, 수정, 커밋, 머지

섹션 1.4.3 의 단계를 밟아왔다면, Bitbucket 이 소스 저장소의 README.rdoc 파일을 자동으로 인식하지 못하고 README 파일이 없다는 메시지를 출력하는 것을 볼 수 있다.(그림 1.16) 이는 rdoc 포맷이 Bitbucket 에서 자동으로 인식하도록 지원할 정도로 많이 사용되지 않는다는 의미다. 사실 저자와 저자가 아는 다른 개발자들은 마크다운을 사용하는 것을 더 선호한다. 이번 섹션에서는 레일스 튜토리얼에 대한 설명을 담고 있는 README 로서 README.rdoc 파일을 README.md 로 변경할

것이다. 이 과정을 통해 저자가 추천하는 Git 으로 브랜치, 수정, 커밋, 머지하는 작업흐름을 알게 될 것이다.13

images/figures/bitbucket_no_readme
그림 1.16: Bitbucket 에서 README 파일이 없다는 것을 알리는 메시지

브랜치

Git 의 장점 중 하나는 브랜치를 만들기 쉽다는 것인데, 이를 통해 원본 소스를 수정하지 않고 소스 저장소의 사본을 만들고 이를 (아마도 실험적인)변경을 할 수 있게 해준다. 대부분의 경우 원본 저장소는 마스터 브랜치가 된다. 새 토픽 브랜치를 만들려면 checkout 명령어에 -b 플래그를 붙이면 된다.

```
$ git checkout -b modify-README
Switched to a new branch 'modify-README'
$ git branch
  master
* modify-README
```

여기서 두번째 명령, git branch 는 로컬 브랜치의 목록을 보여준다. 그리고 별표*는 우리가 작업하고 있는 현재 브랜치라는 표시다. git checkout -b modify-README 명령은 새 브랜치를 생성하고 그 브랜치로 전환하라는 것이고, 이후 modify-README 브랜치 앞에 별표가 표시되는 것에 주목하라. (섹션 1.4 에서 co 별칭을 설정했다면 git co -b modify-README 명령을 사용해도 된다.)

브랜치를 만드는 것의 유용함은 여러 개발자와 협업 할 때 극명하게 나타난다. 14 하지만 이 튜토리얼처럼 혼자 개발 할 때에도 유용하다. 마스터 브랜치는 토픽 브랜치의 변경사항과는 격리되어 관리된다. 그러므로 작업했던 내용이 완전히 엉망이 되더라도 언제든지 작업 내용을 버리고 마스터 브랜치로 체크 아웃한 후 토픽 브랜치를 삭제하여 변경 전으로 되돌아갈 수 있다. 섹션 마지막에 이것도 보여 줄 것이다.

이와 같이 작은 변경을 할 때에 브랜치를 만드는 것이 일반적이라 할 수 없지만, 좋은 습관을 들이자는 맥락에서 보면 아주 좋은 기회라 할 수 있다.

수정

토픽 브랜치를 만든 후 설명을 좀 더 자세하게 기술하도록 수정할 것이다. 저자는 마크다운 마크업 언어를 RDoc 보다 선호한다. 파일 확장자가 .md 인 경우,

Bitbucket 은 이를 보기 좋게 출력해 준다. 그러므로 유닉스 mv (move) 명령어의 Git 버전을 이용해 이름을 변경한다.

```
$ git mv README.rdoc README.md
```

다음에는 README.md 내용을 목록 1.13 과 같이 변경한다.

목록 1.13: 새 README 파일, README.md.

```
# 루비온레일스 튜토리얼: "hello, world!"

이것은 [Michael Hartl](http://www.michaelhartl.com/)의
[*루비온레일스 튜토리얼*](http://www.railstutorial.org/) 첫번째 애플리케이션입니다.
```

커밋

이렇게 변경 한 후, 브랜치의 상태를 확인한다.

```
$ git status
On branch modify-README
Changes to be committed:
  (use "git reset HEAD ..." to unstage)

  renamed:    README.rdoc -> README.md

Changes not staged for commit:
  (use "git add ..." to update what will be committed)
  (use "git checkout -- ..." to discard changes in working directory)

  modified:   README.md
```

여기서 섹션 1.4.1.2 에서 처럼 git add -A 사용 할 수도 있고 git commit 명령에 -a 플래그를 사용하여 기존의 파일중 변경 된 모든 파일을 커밋하게 할 수도 있다. (혹은 git mv 명령으로 파일명을 변경해도 된다.)

```
$ git commit -a -m "Improve the README file"
2 files changed, 5 insertions(+), 243 deletions(-)
delete mode 100644 README.rdoc
create mode 100644 README.md
```

-a 플래그를 사용할 때엔 실수 하지 않도록 주의한다. 마지막 커밋 이후로 파일이 추가 된 경우엔 git add -A 명령으로 추가된 파일을 Git 으로 등록한다.

커밋 메시지를 작성할 때는 현재시제로 작성한다(기술적으로 말하면 명령어법으로 사용한다). Git 모델은 패치의 모음이라 할 수 있다. 이런 관점에서 보면 커밋 메시지를 해당 파일이 무엇을 했는지에 대해 남기기 보다 무엇을 하는지에 대해 적는것이 합당하다. 게다가 이렇게 작성 하는 것이 Git 명령으로 생성되는 커밋 메시지와도 일치하는 것이다. 자세한 사항은 “커밋 메시지 멋지게 작성하기”를 보라.

머지

첫번째 변경작업을 끝냈다면 이를 마스터 브랜치로 머지하여 합칠 수 있다.

```
$ git checkout master
Switched to branch 'master'
$ git merge modify-README
Updating 34f06b7..2c92bef
Fast forward
README.rdoc     | 243 ------------------------------------
README.md       |   5 +
2 files changed, 5 insertions(+), 243 deletions(-)
delete mode 100644 README.rdoc
create mode 100644 README.md
```

Git 의 출력 문구에서 34f06b7 같은 표현이 자주 나타나는 것을 볼 수 있는데 이는 Git 에서 각 저장소를 식별하는 내부 표식이다. 세세한 출력내용이 위의 예제와 완전히 일치하지는 않겠지만 중요한 부분은 위와 동일해야 한다.

변경 사항을 머지하고 나면 git branch -d 명령으로 토픽 브랜치를 삭제하여 브랜치를 정리 한다. 다음 처럼하면 된다.

```
$ git branch -d modify-README
Deleted branch modify-README (was 2c92bef).
```

이 단계는 해도 그만 안해도 그만이다. 사실 토픽 브랜치를 온전히 남겨두는 것이 일반적이다. 그래야만 토픽 브랜치와 마스터 브랜치 양쪽으로 전환 할 수 있고, 작업을 완료 했을때 머지할 수 있게 된다.

앞에서 언급했듯이 토픽 브랜치에서 작업했던 것을 포기해 버릴 수 있다. 이럴 때엔 git branch -D 명령을 사용한다.

# 단지 예시를 위한 것임. 브랜치를 엉망으로 만들지 않기 위해서는 아래와 같이 하지 않도록 한다.
$ git checkout -b topic-branch
$ <really screw up the branch>
$ git add -A
$ git commit -a -m "Major screw up"
$ git checkout master
$ git branch -D topic-branch

-d 플래그를 사용할 때와는 달리, -D 플래그를 사용하면 변경 사항의 머지 여부와 무관하게 브랜치를 삭제한다.

푸시

이제 README 파일을 수정했으므로 이를 Bitbucket 에 푸시하고 결과를 보자. 처음 푸시를 하고 나면 (섹션 1.4.3), 대부분의 시스템에선 origin master 를 생략하고 git push 만 입력해도 된다.

$ git push

섹션 1.4.4.2 에 서 언급했듯이, Bitbucket 은 마크다운으로 작성된 새 파일을 멋지게 출력해준다. (그림 1.17).

images/figures/new_readme_bitbucket
그림 1.17: 마크다운으로 작성하여 개선한 README 파일

1.5 배포하기

지금처럼 개발 초기 단계에서도 레일스 애플리케이션(작업한 내용이 거의 없지만)을 운영 환경으로 배포 할 수 있다. 이 과정은 안해도 되는 부분이지만, 개발 과정에서 최대한 빨리 그리고 자주 배포하면 배포와 관련된 문제를 조기에 쉽게 발견할 수 있게 해준다. 개발 환경에서 큰 작업을 끝낸 이후에만 배포하는 대안을 선택하면 서비스 오픈 시간이 다가올수록 통합하는데 골치아픈 일들이 발생하곤 한다.15

SungHo Choi 6/12/2016 6:04 PM
Comment [8]: 왜에 대한 설명이 너무 짧아보입니다. CI/ CD 에 대한 부연설명 혹은 link 가 간단하게라도 있으면 이해하기 더 좋을 것 같습니다.

레일스 애플리케이션을 배포하는 것은 힘든 과정이었으나 최근 몇 년동안 레일스 배포 생태계에 많은 발전이 있어 지금은 몇 가지 좋은 선택지가 생겼다. Phusion Passenger (아파치나 엔진엑스 16 웹서버 모듈)를 돌릴 수 있는 가상 사설 서버

호스팅 서비스, 엔진야드나 레일스 머신 같은 풀 서비스 배포 회사, 그리고 엔진야드 클라우드, 나인 폴드, 허로쿠 같은 클라우드 개발 환경을 들 수 있다.

저자가 가장 선호하는 배포 방식은 허로쿠를 사용하는 것이다. 허로쿠는 레일스 또는 그 외 웹 애플리케이션을 배포하는데 특화된 호스팅 플랫폼이다. 허로쿠는 소스 코드가 Git 으로 버전 관리되는 경우, 레일스 애플리케이션을 매우 쉽게 배포할 수 있게 해준다. (이것이 섹션 1.4 의 Git 설정을 따라야 하는 또 다른 이유기도 하다.) 이 섹션의 나머지 부분은 첫번째 애플리케이션을 허로쿠에 배포하는 것이 될 것이다. 몇 가지 개념들이 어렵게 느껴 질 수 있지만, 지금은 모든 것을 이해 하지 못한다 해도 걱정하지 말라. 중요한 것은, 이 과정을 수행하면 애플리케이션이 실제 웹으로 배포된다는 것이다.

1.5.1 허로쿠 설정

허로쿠는 PostgreSQL 데이터베이스를 사용한다("포스트그레스큐엘" 이라 발음하며 짧게 "Postgres" 라고도 한다). 그러므로 pg 젬을 운영 환경에 추가하여 레일스가 Postgres 와 연동되도록 한다.17

```
group :production do
  gem 'pg',              '0.17.1'
  gem 'rails_12factor', '0.0.2'
end
```

rails_12factor 젬도 추가 했는데 이는 허로쿠에서 이미지나 스타일 시트 같은 정적 애셋을 서빙할 수 있게 해준다. 수정된 Gemfile 파일은 목록 1.14 와 같다.

목록 1.14: 젬이 추가된 Gemfile.

```
source 'https://rubygems.org'


gem 'rails',         '4.2.0'
gem 'sass-rails',    '5.0.1'
gem 'uglifier',      '2.5.3'
gem 'coffee-rails', '4.1.0'
gem 'jquery-rails', '4.0.3'
gem 'turbolinks',    '2.3.0'
gem 'jbuilder',      '2.2.3'
gem 'sdoc',          '0.4.0', group: :doc
```

```
group :development, :test do
  gem 'sqlite3',      '1.3.9'
  gem 'byebug',       '3.4.0'
  gem 'web-console', '2.0.0.beta3'
  gem 'spring',       '1.1.3'
end

group :production do
  gem 'pg',              '0.17.1'
  gem 'rails_12factor', '0.0.2'
end
```

운영서버로 배포하기 위해 시스템을 준비한다. bundle install 명령을 실행할 때, 운영환경에서 사용할 젬이 로컬 환경에 설치되지 않도록 플래그를 추가한다.(이번 경우에는 pg 와 rails_12factor 가 될 것이다):

```
$ bundle install --without production
```

목록 1.14 에서 추가된 젬들은 운영 환경에서만 사용되어야 하므로 지금 당장 이 명령으로 로컬에 젬을 설치하지 않지만, Gemfile.lock 파일에는 pg 와 rails_12factor 젬이 업데이트 되도록 해야 한다. 다음의 명령으로 변경 사항을 커밋한다.

```
$ git commit -a -m "Update Gemfile.lock for Heroku"
```

다음으로 새 허로쿠 계정을 생성하고 몇가지 설정을 해야 한다. 첫 단계로 허로쿠에 가입한다. 그리고 허로쿠 커맨드라인 클라이언트가 이미 설치되있는지 확인한다.

SungHo Choi 6/12/2016 6:04 PM
Comment [9]: Heroku signup link 가 깨져있습니다.

```
$ heroku version
```

클라우드 통합 개발환경를 사용할 경우 허로쿠 버전 넘버가 보일 경우는 허로쿠 CLI 가 사용가능하다는 것을 의미한다. 그 외의 경우에는 Heroku Toolbelt 로 가서 설치한다.18

허로쿠 커맨드라인 인터페이스가 설치되었다면 heroku 명령어로 로그인하고 SSH 키를 추가한다.

```
$ heroku login
$ heroku keys:add
```

마지막으로 heroku create 명령으로 sample 애플리케이션이 위치할 히로쿠 서버 상의 공간을 생성한다. (목록 1.15).

목록 1.15: Heroku 에 새 애플리케이션 생성.

```
$ heroku create
Creating damp-fortress-5769... done, stack is cedar
http://damp-fortress-5769.herokuapp.com/ | git@heroku.com:damp-fortress-5769.git
Git remote heroku added
```

heroku 명령은 새 애플리케이션을 바로 확인 할 수 있는 서브 도메인을 생성한다. 여기에 아직 아무것도 없으니, 이제 배포를 시작한다.

1.5.2 히로쿠 배포 첫단계

애플리케이션을 배포하는 첫 단계는 히로쿠 마스터 브랜치로 Git 푸시하는 것이다.

```
$ git push heroku master
```

(경고 메시지가 나올 수도 있으나 지금은 무시한다. 이 부분은 섹션 7.5 에서 다루도록 하겠다.)

1.5.3 히로쿠 배포 두번째 단계

사실 두번째 단계는 없다! 이미 배포가 끝난 것이다. 새로 배포한 애플리케이션을 확인하려면 heroku create 을 실행할 때 출력된 주소를 열어본다 (목록 1.15). (클라우드 통합 개발환경 대신 로컬 머신에서 개발하고 있는 경우엔 heroku open 명령을 사용한다.) 그 결과는 그림 1.18 과 같다. 이 페이지는 그림 1.12 과 동일 하지만 이 페이지는 웹에 배포되어 운영 환경으로 동작하는 것이다.

images/figures/heroku_app_hello_world
그림 1.18: 히로쿠에서 동작하는 첫번째 레일스 튜토리얼 애플리케이션.

1.5.4 히로쿠 명령

히로쿠 명령은 방대하지만 이 책에선 간략히 살펴보겠다. 아래와 같이 애플리케이션 이름을 변경할 수 있다.

```
$ heroku rename rails-tutorial-hello
```

위 명령을 그대로 사용하면 안된다. 저자 것이다! 이 부분은 불필요해 보일 수도 있다. 허로쿠가 제공하는 기본 주소로도 충분하기 때문이다. 하지만 애플리케이션 이름을 아래와 같이 랜덤하고 알아보기 힘든 도메인으로 숨기고 싶은 경우 이를 변경 할 수 있다.

hwpcbmze.herokuapp.com
seyjhflo.herokuapp.com
jhyicevg.herokuapp.com

이렇게 하면 주소를 알려준 사람들만 사이트에 접속 할 수 있다. (추가로 루비의 놀라운 간결성의 한 예로 저자가 사용한 랜덤 서브 도메인 생성 코드를 보여주겠다.

('a'..'z').to_a.shuffle[0..7].join

멋지지 않은가.)

허로쿠는 서브도메인 뿐만 아니라 커스텀 도메인 또한 설정할 수 있다. (사실 루비온레일스 튜토리얼 사이트도 허로쿠에서 돌아가고 있다. 이 책을 온라인으로 읽고 있다면 허로쿠에서 호스팅하고 있는 페이지를 보고 있는 것이다.) 커스텀 도메인이나 그 외의 설정에 대해선 허로쿠 문서를 보라.

1.6 결론

이번 장을 거치면서 설치부터 개발환경 구성, 버전 관리와 개발까지 다루었다. 이후로는 레일스로 무엇을 할 수 있는지 1 장 부터 데이터베이스를 사용하는 toy 앱을 만들면서 살펴 보겠다.

여기까지의 진도를 공유하고 싶다면 다음 처럼 트위터나 페이스북 상태를 업데이트 해보면 어떨까?

나는 @railstutorial 로 루비온레일스 공부하고 있어요!
http://www.railstutorial.org/

레일스 튜토리얼 이메일 리스트 1920 에 가입하면 루비온레일스 튜토리얼의 최신 업데이트에 대한 정보나 특별 할인 코드를 받을 수 있다.

1.6.1 이번 장에서 배운 것

- 루비온레일스는 루비 프로그래밍 언어로 작성 된 웹 개발 프레임워크다.
- 미리 구성 된 클라우드 환경에서는 레일스 설치하기, 애플리케이션 생성, 결과 파일에 대한 수정을 쉽게 할 수 있다.

- 레일스는 내장된 커맨드라인 명령 rails 로 새 애플리케이션을 생성하고(rails new) 로컬 서버를 실행했다(rails server).
- 컨트롤러 액션을 추가하고 루트 라우트를 수정하여 “hello, world” 애플리케이션을 만들었다.
- 소스 유실을 막고 협업이 가능하도록 소스 코드를 Git 버전 관리 시스템에 등록하고 이를 Bitbucket 의 비공개 저장소에 푸시하였다.
- 애플리케이션을 허로쿠 운영 환경으로 배포 했다.

1.7 연습문제

- www.railstutorial.org 를 방문하여 레일스 튜토리얼을 구매하면 연습문제에 대한 해답을 구할 수 있다.
- 레일스 튜토리얼 이메일 리스트에 등록하면 보너스로 첫번째 toy 앱에 대한 레일스 튜토리얼 치트시트 카드를 얻게 된다.

1. 목록 1.8 의 hello 액션의 내용을 “hello, world!” 에서 “hola, mundo!” 로 변경해 보라. 보너스 점수: “ ¡Hola, mundo!”에서와 같이 위아래가 뒤집힌 느낌표 문자를 포함해서 레일스가 비아스키(ASCII) 문자를 지원한다는 것을 보여라.(그림 1.19).21
2. 목록 1.8 의 hello 액션처럼 두번째 액션 goodbye 를 추가하고 “hello, world!”을 출력하라. 목록 1.10 의 라우트 파일을 수정하여 루트 라우트가 hello 액션에서 goodbye 액션을 향하도록 변경하라(그림 1.20).

images/figures/hola_mundo

그림 1.19: 루드 라우트를 변경하여 “ ¡Hola, mundo!” 반환하기.

images/figures/goodbye_world

그림 1.20: 루트 라우트를 수정하여 “goodbye, world!”를 출력하기.

루비온레일스 튜토리얼 최신판은 공식 웹 사이트 http://www.railstutorial.org/에서 확인 가능하다. 온라인에서 이 책을 읽고 있다면 최신 업데이트를 보기 위해서는 http://www.railstutorial.org/book 에서 레일스 튜토리얼 책 온라인 버전을 반드시 확인해야 한다. ↑

URI 는 Uniform Resource Identifier 를 의미한다. 이것은 URL, 즉 Uniform Resource Locator 보단 일반적으로 덜 사용된다. 보통 URL 은 “브라우저 주소창에서 볼 수 있는 것”을 의미한다. ↑

http://tryruby.org/ ↑

http://www.railstutorial.org/#help ↑

또한 윈도우 사용자들은 InstallRails 에서 추천한 레일스 인스톨러가 구 버전이고 현재 튜토리얼과는 맞지 않을 수 있다는 점을 염두에 둬야 한다. ↑

예를 들어 foo 라는 함수 정의를 찾는다고 했을 때, 코드 전체에서 "def foo" 를 검색 할 수 있다. ↑

https://c9.io/web/sign-up/free ↑

현재로선, Cloud9 는 이 튜토리얼과 호환되지 않는 구버전의 레일스를 제공한다. 따라서 직접 설치 해야 한다. ↑

표 3.1 처럼, install 을 생략할 수 있는데, bundle 명령 자체가 bundle install 의 별칭이므로 이 명령만 입력하더라도 동일하게 동작한다. ↑

보통 웹사이트는 80 포트를 사용한다. 이렇게 설정하려면 특별한 권한이 필요하므로 개발 서버에서는 사용에 제약이 적은 높은 번호의 포트를 사용하는 것이 관례다. ↑

위에서 "Ctrl-C" 를 입력하려고 Shift 키를 누르지 않고 "Ctrl-c" 만 눌러도 된다. 어떤 이유에서인지 늘 "Ctrl-C" 로 표기한다. ↑

이 튜토리얼에서 전혀 수정할 필요가 없는 부분이지만, .gitignore 파일에 규칙을 추가하는 예는 섹션 3.7.3 에서 볼 수 있는데 이것은 섹션 3.7 에서 고급 테스팅을 위한 셋업 옵션의 일부분이다. ↑

Git 저장소를 시각화하여 사용하려면 Atlassian 사의 SourceTree 앱을 검색하라. ↑

Pro Git 에서 브랜칭 장을 참조하라 ↑

레일스 튜토리얼의 예제 애플리케이션이 문제가 있는 것은 아니지만, 독자의 앱을 공개하기에 너무 이르다 생각한다면 몇 가지 옵션이 있다. 섹션 1.5.4 에서 알아보자. ↑

"엔진엑스"라 발음해요. ↑

일반적으로 개발과 운영 환경을 최대한 일치시키는 것이 좋다. 이 튜토리얼에서는 목적상 SQLite 는 로컬에서만 사용하고 PostgreSQL 은 운영 환경에서 사용할 것이다. 자세한 설명은 섹션 3.1 을 참고하라. ↑

https://toolbelt.heroku.com/ ↑

레일스 튜토리얼 메일링 리스트 ↑

http://www.railstutorial.org/#email ↑

편집기에서 "invalid multibyte character"라는 메시지를 출력할 수도 있지만, 이는 여기서 다루지 않겠다. 해결 방법을 알고자 한다면 에러 메시지를 구글링 할 수 있다. ↑

Powered by RORLAB®