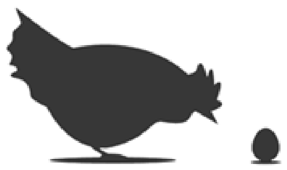
Big Fat Rails



by Mitch Guthrie

February 20, 2012

Contents

Chapter 1

What’s This Book About?

The purpose of this book is to teach someone new to Ruby on Rails how to get stuff done and start developing their own applications quickly. A focus will be on learning topics precept upon precept in small digestable steps.

As the book proceeds the reader will build upon existing knowledge to grow deeper in understanding and connect not just the “how” of the process, but the “why” as well.

Underlying ideas will be covered like MVC, REST, and how to organize code so that it makes sense to you and those who will work with you.

There are a lot of things to learn when developing an application by current standards such as TDD, BDD, proper MVC and so on. It is here that we run into a chicken and egg problem that is inherit in learning any complicated subject: the theory is often hard to understand separate from seeing it in practice, and seeing it in practice can then be difficult to understand if you don’t have a full grasp of the theory. (Feel free to read that again, I had to.)

Consider a math teacher who would write an equation on the board then say something completely irksome like “and if we factor and reduce this equation we get this”. This is great if you know the process of factoring and reducing but if you don’t that final equation can seem almost magical.

What this means to the reader is that sometimes we may progress down a path in a fashion that seems contrary to what is considered good practice but in fact is only done so we can understand what a “good practice” is and why we should do it that way.

Ruby on Rails does a lot of "factor and reduce" to make development easier for those who know the conventions. My goal with this book is to fill in some of the factor and reduce material so you, the reader, have a firm grasp on what's going on.

That is the last analogy I'll discuss regarding math. I promise.

1.1 This Book Is A Work In Progress!!

This book is currently being developed. Every ebook should have a version number. Things are going to be changing a lot while this book is in development. The latest and greatest can always be found at `http://www.bigfatrails.com`.

1.2 What Should I Already Know?

You should have experience with HTML/XHTML, CSS, Javascript, and a basic understanding of how the web works. A basic proficiency with Ruby is a must and there a plenty of tutorials to get you up to speed.

1.3 About The Author

My name is Mitch Guthrie and I am a web developer. I'm just a guy. I don't really have any fancy pedigree, I'm not formally trained, but I've made it my goal to persue quality and depth in my day to day. I would call myself a "secondary teacher". My desire in teaching others can be summed up with the following quote [1]:

> [The secondary teacher] should regard himself as learning from the masters along with his [students]. He should not act as if he were a primary teacher, using a great book as if it were just another textbook of the sort one of his colleagues might write. He should not masquerade as one who knows and can teach by virtue of his original discoveries... The primary sources of his own knowledge should be the primary sources of learning for his students, and such a teacher functions honestly only

[1] Mortimer Adler, How to Read a Book (New York: Simon and Schuster, 1940), p. 60.

> if he does not aggrandize himself by coming between the great books and their... readers. He should not "come between" as a nonconductor, but he should come between as a mediator—as one who helps the less competent make more effective contacts with the best minds. [2]

I'm not perfect. Things will change. What was once considered good programming practice is now the albatross around the neck of the programmer [3]. The idea isn't to get everything perfect from here on out. The idea is to improve consistently over time.

You can contact me at mitch@bigfatrails.com or follow me on twitter: @mentalbrew `http://twitter.com/mentalbrew`.

1.4 Conventions

All source code will be in a mono font and indented like this:

```ruby
class Example
  def initialize(greeting=nil)
    @greeting = greeting
  end
end
```

Terminal output will look similar:

```
total 24
drwxr-xr-x 7 mentalbrew mentalbrew 4096 2011-08-02 12:02 build
-rw-r--r-- 1 mentalbrew mentalbrew 4519 2011-08-02 08:52 make.bat
-rw-r--r-- 1 mentalbrew mentalbrew 4602 2011-08-02 08:52 Makefile
drwxr-xr-x 4 mentalbrew mentalbrew 4096 2011-08-02 08:52 source
```

NOTE A note will be used point out something interesting or significant.

[2] I discovered this quote from watching "Why You Don't Get Mock Objects by Gregory Moeck" - `http://www.youtube.com/watch?v=R9FOchgTtLM`

[3] http://en.wikipedia.org/wiki/The_Rime_of_the_Ancient_Mariner

WARNING A warning will serve to point out something you should know regarding the topic that could be a gotcha.

You can find footnotes like this [4] at the bottom of the page.

1.5 License

1.6 Credits

The chicken and egg logo are from the artwork titled Paradox by Carlos Navas. Used with permission.

The music for the screencasts is Slumlord by Lo Tag Blanco and licensed under Creative Commons.

1.7 Software Versions

This book uses Rails 3.2 and Ruby 1.9.3. I'm developing on both Mac OS X Lion and Ubuntu 11.10.

To avoid getting bogged down in potential design issues and since this book is not covering web design we will be using the Twitter Bootstrap [5] framework to quickly get are app up and looking decent.

[4] Related information will be available in the footnotes.
[5] `http://twitter.github.com/bootstrap/`

1.8 FAQ

- Will this make me a Rails guru?
  - Yes. All you need to do is read this book as well as every other Rails book, and practice everyday for over 10,000 hours.
- Urban Dictionary says that a "fat rail" is a thick line of cocaine. Does this book have anything to do with cocaine?
  - No, this book has nothing to do with cocaine but if it perhaps garners interest in the coveted recovering addict demographic then I think they should be able to learn Rails if they want. If I were to exclude any book title that matches something in Urban Dictionary then this would simply be titled "A book".
- I don't like your writing style. Can you change it?
  - I don't like your reading style. No.

1.9 Disclaimer

The author assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein. Should this book result in mental trauma, physical damage of computer equipment, or a bill of divorcement from spouse that would be really sad but not the author's fault.

Chapter 2

Tools Of The Trade

The first thing any developer needs to do is get their environment configured and running. This can be difficult for the new user because often the perfect setup other programmers devise has been crafted through years of working with the technology.

It's my goal to get a user started in the right direction with, what I would consider, the most useful setup there is.

2.1 Requirements

It is greatly encouraged that you be using a computer with either Mac OS X or some flavor of Linux. In this book I will cover instructions for OS X Lion and Ubuntu 11.10.

2.2 What About Windows?

Ruby and the Ruby on Rails community cut its teeth in the Unix/Linux tradition and as such those operating systems are better suited for this kind of work. Huge progress is being made however of bringing Windows into the fold but there's still a lot of work to be done.

Many people sucessfully develop on a Windows platform for Ruby on Rails but being a beginner I would highly advise against it at this time. You may find yourself struggling with a problem that isn't your fault but a quirk of simply trying to develop under Windows. For the brave, however, I say go ahead and make it happen. I'm sure you'll learn a ton.

2.3 Web Browsers

When developing a web application it is important to test your work across all the browsers your users might be using. Since we are going to be developing for our own purposes I've chosen Google's Chrome (`https://www.google.com/chrome`) as the browser of choice for development and demonstration. Use whatever browser you are more comfortable with but beware if things deviate from what is shown. I utilize Firefox, Chrome, and Safari often. Internet Explorer is the venereal disease of web development; it is my advice to steer clear of it.

2.4 Programming Editor Or IDE?

If you've done programming in another language you are probably already familiar with a specific programming editor or integrated development environment (IDE). Rails is editor agnostic so you're really free to choose what you want. There are several which are free and some that you'll need to purchase a license for. I will say however, that many good Rails developers have a very strong aversion to using an IDE.

Here's a few good editors to choose from for Rails development:

Vim Vim usage is very strong among many in the Rails community and there are a lot of plugins that can help your Rails development drastically. Vim is cross platform and works on Linux, Unix, Mac OS X, and Windows. Vim is its very own beast though and can take a lot of time before you can be moving at a good pace with it. Those who stick with it are usually wicked-fast at editing files and managing their environment. You can find Vim at http://www.vim.org/ as well as MacVim here: `http://code.google.com/p/macvim/`

Redcar Editor This is a very versatile editor, itself written in Ruby running on Java. It has a great amount of features and is open source. Development is fast moving and new features are added all the time. `http://redcareditor.com`. Redcar runs on Linux, Mac, and Windows.

TextMate This is probably the most well known editor and used by many Rails developers. TextMate has a lot of timesaving features like snippets and Goto file. It costs around $58 and only works with Mac. `http://macromates.com`

SublimeText 2 This is the new editor on the block and functions very similar to Redcar and Textmate. It works on Linux, Mac, and Windows. The development is very fast paced with new releases often and has some interesting features of its own. Cost: $59. `http://www.sublimetext.com`

2.5 IDE

There are a few IDEs that are worth mentioning and have a following in the Rails community:

Netbeans An open source IDE with support for Ruby on Rails. Runs on Linux, Mac, and Windows. `http://netbeans.org`

RubyMine A commercial IDE for Rails development that runs on Linux, Mac, or Windows. Cost starts at $69 dollars and can be found at `http://www.jetbrains.com/ruby`

Aptana Studio Cross platform, open source IDE built off of the Eclipse IDE. `http://www.aptana.com`

2.6 Closing Comments Regarding YOUR Editor

Use whatever works best for you. The Rails community by nature is very strongly opinionated and stubborn at times. No matter which of the previous products you choose to do development in you will most likely catch some crap for it. The main thing is to find something that you can use to get stuff done. Screw what people think.

2.7 Managing Your Source Code with Git

This book will utilize and encourage the use of Git (`http://git-scm.com`). Git is a distributed version control system (DVCS). That's basically a really fancy way of saying that it will keep your source code in order and make it easier to work with others on your team. There is a chapter on Git covering the basics. You can get Git for Linux, Mac, and Windows.

2.8 Managing Your Ruby Versions With RVM

The Ruby Version Manager (RVM) can be found at (`http://rvm.beginrescueend.com`) and provides a way to manage multiple Ruby installations for testing and development. Operating systems are notorious for shipping very old versions of Ruby. By using RVM you can have the latest and greatest version(s) of Ruby at your fingertips with very little overhead. The instructions are great for getting it installed and RVM allows you to play with different implementations of Ruby without screwing up your system.

Chapter 3

Setting Up Your Environment

This chapter covers what it takes to get up and running in either Mac OS Lion or Ubuntu 11.10. Feel free to skip to the relevant section for your operating system.

3.1 Ubuntu

I'm going to be using the latest version of Ubuntu at the time of this writing (11.10) to get started on our way to Rails development nirvana. Much of this will also work for previous versions of Ubuntu.

To get started the first thing you will need to do is get Ubuntu installed. There is a lot of documentation out there on how to do this so I won't cover it here. If you find you run into problems hit the interwebs and see if the solution is out there. If you cannot seem to get Ubuntu installed correctly there's a good chance you may have a socially awkward cousin or something that can do it for you.

3.1.1 Get Git

Git is a source code management tool for keeping track of changes in your code base over time and makes collaborating with others an actual enjoyable process. Git is required for RVM so lets install it now. I will go over git later but for now to install Git in Ubuntu just type:

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

3.1.2 Ruby Version Manager (RVM)

```
sudo apt-get install curl

bash -s stable < <(curl -s \
  https://raw.github.com/wayneeseguin/rvm/master/binscripts/rvm-installer)

echo 'export rvm_pretty_print_flag=1' >> ~/.rvmrc

source ~/.profile
```

Running `rvm requirements` on your machine should tell you what system packages you should install that are necessary for RVM to build Ruby.

The following is what is suggested for Ubuntu 11.10:

```
sudo apt-get install build-essential openssl libreadline6 libreadline6-dev \
curl git-core zlib1g zlib1g-dev libssl-dev libyaml-dev libsqlite3-0 \
libsqlite3-dev sqlite3 libxml2-dev libxslt-dev autoconf libc6-dev ncurses-dev \
  automake libtool bison subversion
```

RVM should now be installed and configured correctly. To verify simply type:

```
rvm list
```

It should list currently installed Ruby versions which at this moment should be nothing. Let's get that installed.

In this book we will be using Ruby 1.9.3. Currently the Ruby community is making a transition from Ruby 1.8.7 which is most likely the default if you were to install this through your system package manager. Since, I assume you will be creating green-field apps [1] from here out you might as well start off on the latest and greatest.

To install Ruby 1.9.3 run the following:

```
rvm install 1.9.3
```

[1]A "green-field" app is something you write from scratch and have full control over its development. Unlike and existing app where you have to deal with legacy code.

RVM will then download the latest version of the Ruby 1.9.3 source code, extract, configure, and compile it for installation. The compilation process may take awhile.

Once RVM is done you should be able to type:

```
rvm list
```

This will give a list of all of your installed ruby versions. Since you have a brand new install it should look like this:

```
rvm rubies
    ruby-1.9.3-p0 [ x86_64 ]
```

Your version and the bracketed platform numbers may be different but what's important is that you have ruby-1.9.3 installed.

To tell RVM you would like to use ruby-1.9.3 just type:

```
rvm use 1.9.3
```

Now if you do an 'rvm list' there will be an arrow pointing to ruby-1.9.3 letting you know which ruby you're running at the moment. To test and make sure you have the correct ruby type:

```
ruby -v
```

This will run the ruby intrepreter and give you it's version number. You are now good to go.

To make sure this will stay your default ruby intrepreter you can type:

```
rvm 1.9.3 --default
```

Now, when working in a new terminal RVM will always default to ruby-1.9.3.

3.2 Mac OS X

To get started developing on Mac OS X one of the best tools for installing needed software is Homebrew (`http://mxcl.github.com/homebrew/`). Homebrew uses recipes that it can use to download, compile, and install software. These recipes can be created easily and many people have contributed. This makes a decent alternative to what Ubuntu has in apt-get.

Homebrew requires Xcode to work properly for compiling the packages from recipes. While there are currently other projects out there that can compile some recipes require Xcode. You can obtain Xcode through the App Store.

Once you have Xcode installed you can then run the command to install and configure Homebrew:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.github.com/gist/323731)"
```

You will be asked questions during the install. Once everything is complete you should be able to use homebrew to install needed libraries and packages.

3.2.1 Install Git

```
brew install git
```

3.2.2 Ruby Version Manager (RVM)

```
brew install curl
```

```
bash -s stable < <(curl -s \
  https://raw.github.com/wayneeseguin/rvm/master/binscripts/rvm-installer)
```

```
echo 'export rvm_pretty_print_flag=1' >> ~/.rvmrc
```

```
source ~/.profile
```

Running `rvm requirements` on your machine should tell you what system packages you should install that are necessary for RVM to build Ruby.

Currently RVM states that if you wish to install different versions of Ruby other than 1.9.3 you need to have Xcode 4.1. Since this book is only going to require 1.9.3 you should have installed Xcode 4.2.

RVM should now be installed and configured correctly. To verify simply type:

```
rvm list
```

It should list currently installed Ruby versions which at this moment should be nothing. Let's get that installed.

In this book we will be using Ruby 1.9.3. Currently the Ruby community is making a transition from Ruby 1.8.7 which is most likely the default if you were to install this through your system package manager. Since, I assume you will be creating green-field apps [#fn1]_ from here out you might as well start off on the latest and greatest.

To install Ruby 1.9.3 run the following:

```
rvm install 1.9.3
```

RVM will then download the latest version of the Ruby 1.9.3 source code, extract, configure, and compile it for installation. The compilation process may take awhile.

Once RVM is done you should be able to type:

```
rvm list
```

This will give a list of all of your installed ruby versions. Since you have a brand new install it should look like this:

```
rvm rubies
    ruby-1.9.3-p0 [ x86_64 ]
```

Your version and the bracketed platform numbers may be different but what's important is that you have ruby-1.9.3 installed.

To tell RVM you would like to use ruby-1.9.3 just type:

```
    rvm use 1.9.3
```

Now if you do an ‘rvm list’ there will be an arrow pointing to ruby-1.9.3 letting you know which ruby you’re running at the moment. To test and make sure you have the correct ruby type:

```
    ruby -v
```

This will run the ruby intrepreter and give you it’s version number. You are now good to go.

To make sure this will stay your default ruby intrepreter you can type:

```
    rvm 1.9.3 --default
```

Now, when working in a new terminal RVM will always default to ruby-1.9.3.

Chapter 4

Git Bootcamp

The purpose of this book is to give a good grounding in web development using Rails and more importantly, getting started in the right direction.

While this chapter really has nothing to do with development in Rails it does have a lot to do with programming etiquette. Every programmer should use source code management. Every programmer. No excuses.

It's good for you (the programmer), your team, and anybody else who has to come behind you when leave because your on-the-side startup "pivoted" and "took off" while reaching your "core market demographic".

4.1 What is Git?

Git is a source code management system for keeping track of changes in your code base. As you develop your application you will find yourself in a good place to make a backup copy "just in case" or, maybe you are planning on doing a big change that you might need to reverse or undo.

Git allows you to take snapshots of your code at any specific point and provide a comment or detailed information about what's going on. You can then use Git to roll back or move forward among these snapshots.

Git also has the ability to tag a particular snapshot or commit so you can always refer to it. Developers use tags to mark a specific version or release of the software they are working on.

One very important aspect of Git is the ability to branch a codebase and make whatever changes you wish without disturbing the master copy of the code. If you like the work you've done in a branch you can simply merge back into the main repo. If you don't like it there's nothing stopping you from deleting it for good.

Branches allow a developer to work on a new version of their app without effecting the current working version. You can be working in your latest whiz-bang feature and switch back to the master branch and quickly fix a bug.

The mighty feature of Git is being able to work offline. Previously, older SCM's were only able to make a commit if you could connect to the server it was on and create your changes. In Git, you have a fully working copy of the repository that you can take with you anywhere you want to go. You can continue to make commits and move forward with your development. When you get a chance to connect to your server it will simply push your commits if there are no current changes to the repository.

4.2 Configure Git

The first thing a person needs to do is get Git configured for use. If you haven't already done so please install Git as noted in the previous chapter.

Git likes to have a name and email address associated with every commit. You can set these easily with the following:

```
git config --global user.name ”Your Name”
git config --global user.email you@example.com
git config --global color.ui auto
```

Here we simply tell Git that the user.name should be whatever name is and user.email should be whatever email you want associated with the work you're doing.

In Linux/Mac this will create a file in your home directory called .gitconfig. This file is what Git uses to keep settings.

The third line in the previous command simply turns on console coloring if it's available. This is a nice feature to have as it makes the output of Git easy to quickly parse for what you're looking for.

4.3 Creating

Once you have things configured you need to create or clone a Git repo to work with. In this chapter we'll simply create a repo so you can get the basics of Git.

Starting with creating a simple directory:

```
mkdir mydirectory
```

Then, we'll move into the directory and initialize a basic Git repository:

```
cd mydirectory
git init
```

You now have a fully working Git repo. From here we can then start creating files and directories. This would be the source code to your application. For now, create three files in whatever editor you want to called `file1.txt`, `file2.txt`, and `file3.txt`. Now we need to make Git aware that it should track changes in these files. To see how your Git index looks type the following:

```
git status
```

The output should look something like this:

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   file1.txt
#   file2.txt
#   file3.txt
nothing added to commit but untracked files present (use "git add" to track)
```

If you've enabled color you should see the files listed in red. This is Git's way of telling you they either needed to be added to the repo or have changed since they were last committed.

To add these file so Git will track them simply type:

```
git add .
```

This is a quick way of telling Git to add any untracked or changed files in this current directory (which is what the . means).

Now your git status about should be show the following files in green:

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use ”git rm --cached <file>...” to unstage)
#
#   new file:   file1.txt
#   new file:   file2.txt
#   new file:   file3.txt
#
```

This shows that Git is now ready to commit these changes to the repository. So, let's get this committed and tracked:

```
git commit -m ”Added three new files”
```

You should see something similar to this afterward:

```
[master (root-commit) 9ad0290] Added three new files
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
 create mode 100644 file2.txt
 create mode 100644 file3.txt
```

What this will do is add all of the staged or "green" files to the repository with a commit message that says "Added three new files". You now have files that are being tracked in Git. To see a log of all the past commits and their messages simply type:

```
git log
```

You'll see something like this:

```
commit 9ad0290f44f3601554ab08f717f096f50d95b01d
Author: Mitch Guthrie <mitch@mentalbrew.com>
Date:   Sat Jul 30 22:41:13 2011 -0700

    Added three new files
```

So far so good.

4.4 Making More Changes

We now have our very first commit in a git repository. You should be aware that this is local on your machine. It isn't stored or hosted anywhere else. Then entire functionality of this new git repo is stored in the root of the project directory under `.git/`. Do NOT delete this directory as you will then essentially destroy all your past commits and repository information.

Open `file2.txt` and enter some text then save it. If you go back to your terminal and type:

```
git status
```

You should see the following output:

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
```

```
  #  (use "git checkout -- <file>..." to discard changes in working directory)
    #
    #   modified:   file2.txt
    #
    no changes added to commit (use "git add" and/or "git commit -a")
```

What this is telling you is that git has recognized that the contents of `file2.txt` have been modified. If you wish to keep this change you can simply add the file to the index, and then commit like so:

```
git add file2.txt
```

```
git commit -m "Updated file2.txt"
```

Now when you run `git log` to see past commits you should see something like this:

```
commit 166962bfe60acdbf9a63db4ecf757ddd4cf419bd
Author: Mitch Guthrie <mitch@mentalbrew.com>
Date:   Mon Feb 20 09:08:38 2012 -0800

    Updated file2.txt

commit daeb73cd46f93b468f805e361c018f39f89d7381
Author: Mitch Guthrie <mitch@mentalbrew.com>
Date:   Mon Feb 20 08:54:51 2012 -0800

    Added three new files
```

Now you can track your past commits using `git log` and can always checkout old versions of your code.

4.5 Stepping Back In Time

If you wish to see an old version of your code you can get the commit hash and tell git you just want to see that version.

When you run `git log` and see past commits you should be aware that your hashes will be different than mine. If you create to repositories that are exactly the same change for change, and commit for commit, those hashes will be different as well. The theoretical realities aside, EVERY commit in a git repo will be unique and most likely completely different from another repo. If you are wondering why my output looks different than yours this is why.

To checkout a specific commit in your history simply type:

```
    git checkout <commit>
```

Where `#commit` is the first few letters of the commit hash. You don't need to type the whole thing.

To get back to your most current version you can always type:

```
    git checkout master
```

Master is the default most recent version of your repository.

4.6 Reversing Changes

Let's say you are working on a big change and about half way through it you realize it's not going to work and you would like to revert back to the original version you were at before starting.

You can get a list of the changed files using `git status` and then to restore them to their previous versions simply type:

```
    git checkout <filename>
```

This will restore that file to its previous version.

4.7 Branching

Rather than having to undo the changes on multiple files it is easier if you branch before doing work on a big feature then merge it back into master when everything is how you want it.

Branching, basically takes a copy of the repository at a certain commit and allows you to work independently on it until you either decide to merge it back into master or simply delete the entire thing.

First you create a branch, then check it out to work on it.

```
git branch awesome
git checkout awesome
```

Or you can shortcut the process by branching and checking out at the same time:

```
git checkout -b awesome
```

Now you can continue working that specific branch without effecting the rest of your repository. You can always find out what branch you are on by viewing `git status`:

```
git status
```

```
# On branch awesome
nothing to commit (working directory clean)
```

Be aware that any new files you create, any files you change or remove are only applicable to this branch and not to anything else outside it including `master`. ## Merging

If you have all of your changes commit in your new branch and you wish to change back to master you can type:

```
git checkout master
```

Now, if you wish to see what branches you have created:

```
git branch
```

Then, if you wish to update your master branch with all of the work you did in the `awesome` branch you can type:

```
git merge awesome
```

Make sure you are in the branch you want your changes to go to first. So, if we want our `awesome` branch merged into `master` we need to be in `master`. If you are looking to pull in the most recent changes in `master` into your local branch `awesome` you will need to be in the `awesome` branch first.

4.8 Tagging

Sometimes, it's beneficial to mark a specific version of the repo with a name. In git this is referred to as a tag and is usually created by developers to mark a specific version of the code base. A tag, really is nothing more than giving a name to a specific commit. If the current version of your master copy was something that you wish to be marked as 'v1.0' then you can do this:

```
git tag -a v1.0 -m 'version 1.0'
```

If you wish to ever checkout out this version of the repo specify that tag name like so:

```
git checkout v1.0
```

4.9 Cloning

In, git, when we make another complete copy of another repository it is called cloning. To clone another repo all you have to do is:

```
git clone <path to repository>
```

You now have your very own working copy of the repository.

4.10 Workflow

Like any development tool, one of the first things you need to do is figure out a workflow that assists in what you need to be doing. I'll quickly walk through a very simple and easy to use workflow for git. I'm assuming you are the only person really utilizing this repository. It should work for groups just the same but often, when more players are involved, you tweak the workflow a bit.

The basic workflow I use when developing is the following:

1. **Create the repo** - Obviously this comes first. Create the repo. Get everything in it you need to work on your project and create your first commit.

2. **Skeleton** - Sometimes it helps to simply get a skeleton of your app going without worrying about the workflow just yet. Make sure things are at a good base before you start diving in to create features. Once you have this solid foundation, commit this and move on to the next step.

3. **Branch for a feature or bug** - Create a branch from master. From here on out master will be your stable version of the code. Everything else you do should be done in a branch then merged into master when it's solid.

4. **Merge with master** - Once your branch is stable, switch to `master` and merge it back in. If you have changes that were added to master after you branched make sure you pull those FROM `master` TO your branch first. That way if there is an issue you can solve it there and not on the main `master`.

5. **Delete old branch** - If you have merged a branch into `master` or decided you don't need to keep the code, just delete the branch. `git branch -d <branchname>` should do the trick.

These are basic steps. You can get a lot more tricky with things if you want but this serves a basic workflow that should save your bacon [1] if necessary.

[1] Or perhaps something that's more kosher

4.11 Wrapping Up

There's a lot to Git. Certainly more than I can cover in a brief overview. I would highly encourage you to learn more about Git as you're literally putting all your eggs in its basket [2]. The following are a list of resources I highly encourage the new Git user to study:

- Pro Git `http://progit.org/`: Basically the cat's meow of Git books. Easy to read and Scott Chacon has provided many pretty little graphs and what not.
- The Git Community Book `http://book.git-scm.com/`: A great resource provided by the community, for the community.
- Git Immersion `http://gitimmersion.com/`: Pretty much exactly like it sounds.

[2]This is a disturbing image for some reason

Chapter 5

Beginning Ruby on Rails

The first aspect of getting started with Ruby on Rails is installing it. We've built up quite the little development process here by using RVM for managing our ruby versions. Now, we are going to be using RubyGems which ships with ruby 1.9.3.

5.1 Installation

To install Rails using RubyGems simply type:

```
    gem install rails --no-ri --no-rdoc
```

This will then go and fetch the most recent development Rails gems and all of the other gem dependencies needed to make Rails function. This will be a long list and may take awhile depending on the speed of your computer and internet access.

The `gem` command is used to install ruby packages and their dependecies. If you know that name of the package you wish to install `gem install <packagename>` will do that for you. Here we specified rails. There are a few flags that we listed after our command which do the following:

–no-ri This will tell rubygems not to install the ruby information regarding each gem. You can have this if you want but it does slow down the install process quite a bit.

–no-rdoc Similar to `--no-ri` this will tell ruby not to install system documentation for each gem. Omitting this makes the installation take longer.

NOTE If you wish to have the –no-ri and –no-rdoc options by default simply create a `.gemrc` file in the root of your home directory and add this line:

```
gem: --no-rdoc --no-ri
```

The following is a brief snippet of what you should see when installing Rails for the first time:

```
Fetching: i18n-0.6.0.gem (100%)
Fetching: multi_json-1.0.4.gem (100%)
... (omitted)
Fetching: json-1.6.5.gem (100%)
Building native extensions.  This could take a while...
Fetching: rdoc-3.12.gem (100%)
Fetching: railties-3.2.1.gem (100%)
Fetching: rails-3.2.1.gem (100%)
Successfully installed i18n-0.6.0
... (omitted)
Successfully installed rails-3.2.1
30 gems installed
```

You now have the most up-to-date version of Rails installed on your system and ready to go. Now that we have Rails installed it's time to create a brand new Rails project.

5.2 A New Rails Project

We are going to create a new project named **Bankroll**. To create this new Rails project you can type the following command:

```
rails new bankroll --skip-bundle
```

This will create a bunch of directories and files for your new Rails project as shown below. The `--skip-bundle` flag tells Rails not to immediately install dependencies for you project.

```
create
create  README.rdoc
... (omitted)
create  vendor/assets/stylesheets/.gitkeep
create  vendor/plugins
create  vendor/plugins/.gitkeep
```

Once that command is done you'll want to change into the project directory like so:

```
cd bankroll
```

The then install all of the necessary packages to get your application up and running and specify that you want binary stubs using the `--binstubs` flag:

```
bundle install --binstubs
```

The following is an edited version of what you should see when bundle is installing dependencies:

```
Fetching source index for https://rubygems.org/
Using rake (0.9.2.2)
... (omitted)
Installing sass-rails (3.2.4)
Installing sqlite3 (1.3.5) with native extensions
Installing uglifier (1.2.3)
Your bundle is complete! Use ‘bundle show [gemname]‘ to see where a
bundled gem is installed.
```

So, what the heck happened? Well, for starters you initiated the `rails new` command which tells the Rails command line utility that you want to create a new project from scratch. What this basically does is create a directory named after the project you specified (in our case it was 'bankroll'). Then in that directory it creates all the files necessary for a vanilla install of Rails.

In the output above, you see where it says "create" over and over. It is just going through the process of creating the files and directories necessary for Rails to function. These files and directories are the basis of your new web application and any Rails app you work on will be organized in this fashion.

Once Rails gets all the files in needs in place you can then run the command `bundle install`. This command is used to pull all of the necessary dependencies or gems that Rails needs to work on your computer. Once the `bundle install` command is complete you can now view the new application directory.

5.3 Finding Your Way Around

The following is a screenshot of the directoy that Rails created. The structure of this project folder will be the same across every Rails application. Rails stresses convention over configuration in everything and as an effect everything in Rails has a place where it belongs.

As you grow in your knowledge of Rails you will quickly become comfortable in where to look for things and how to get around. We will cover the directory structure of Rails in a later chapter. This chapter is just to get things up and running so lets move ahead.

5.4 Fire It Up

To see the beauty that is your future application it's best we can see it in action. Now, if you come from a background in PHP or or Perl you may be used to depending on Apache or the like to view and develop your application. This is not necessary in Rails because it comes with it's very own development web server that you can use to run things quickly and locally for your work. To get it started run the following in the root of your project directory:

```
bundle exec rails server
```

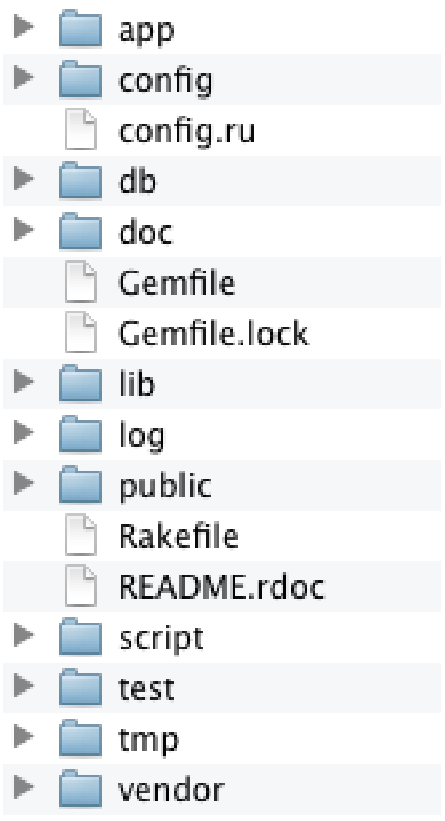


Figure 5.1: Rails Directory Listing

WARNING If your terminal vomits up an error about `'autodetect': Could not find a JavaScript runtime` then skip to the next section for the fix and return back to here.

You should see the following output letting you know that the local webrick server is started and awaiting connections on http://localhost:3000:

```
 => Booting WEBrick
=> Rails 3.2.1 application starting in development on http://0.0.0.0:3000
 => Call with -d to detach
 => Ctrl-C to shutdown server
 [2012-02-06 17:21:31] INFO  WEBrick 1.3.1
[2012-02-06 17:21:31] INFO  ruby 1.9.3 (2011-10-30) [x86_64-darwin11.2.0]
[2012-02-06 17:21:31] INFO  WEBrick::HTTPServer#start: pid=8426 port=3000
```

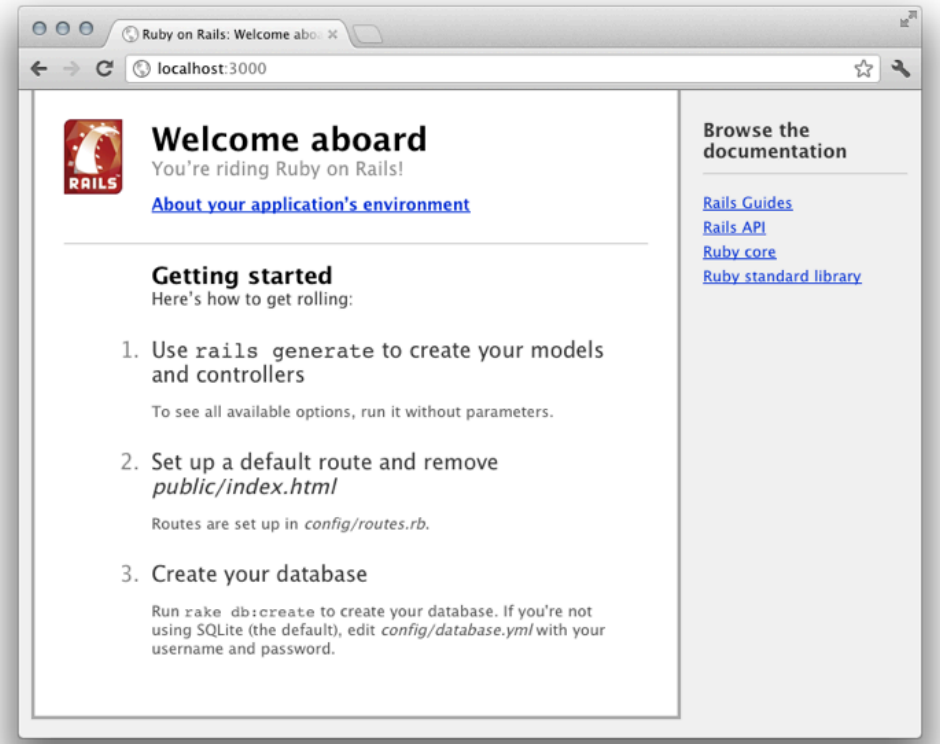
Now, if you open your web browser and visit `http://localhost:3000` you should see the following Rails default page:

If you click on the "About your application's environment" you should see information regarding to your local installation of Rails.

5.5 Adding A Required Gem

If you tried to follow instructions only to find that when you ran `bundle exec rails server` your terminal vomitted out something similar to the following it's because we need to add one more thing for your specific OS.

```
/home/mentalbrew/.rvm/gems/ruby-1.9.3-p0/gems/execjs-1.3.0/lib/execjs
 /runtimes.rb:50:in ‘autodetect’:
Could not find a JavaScript runtime. See https://github.com/sstephenson
 /execjs for a list of
 available runtimes. (ExecJS::RuntimeUnavailable)
from /home/mentalbrew/.rvm/gems/ruby-1.9.3-p0/gems/execjs-1.3.0/lib/
 execjs.rb:5:in ‘<module:ExecJS>’
 ... (omitted)
from /home/mentalbrew/.rvm/gems/ruby-1.9.3-p0/gems/railties-3.2.1/lib/
```



Welcome aboard
You're riding Ruby on Rails!
About your application's environment

Getting started
Here's how to get rolling:

1. Use `rails generate` to create your models and controllers

   To see all available options, run it without parameters.

2. Set up a default route and remove *public/index.html*

   Routes are set up in *config/routes.rb*.

3. Create your database

   Run `rake db:create` to create your database. If you're not using SQLite (the default), edit *config/database.yml* with your username and password.

Browse the documentation
Rails Guides
Rails API
Ruby core
Ruby standard library

Figure 5.2: Default Start Page

```
rails/commands.rb:50...
from script/rails:6:in ‘require’
from script/rails:6:in ‘<main>’
```

What we need to do is open the file under your project called `Gemfile` and add the following line right underneath the line that reads `gem 'rails', '3.2.1'`:

```
gem 'therubyracer'
```

Save and run the `bundle install` command. What this will do is add a needed component that is not installed with Rails by default. If you're on a Mac you would not have noticed this as it comes with the needed component but if you are using Ubuntu or another Linux distribution then this is required.

5.6 Using Binstubs

If you notice, we added a flag to our first `bundle install` command that was `--binstubs`. Because Rails uses the Bundler library `http://gembundler.com/` to manage dependencies you need to always run `bundle exec` before every command to ensure it gets run in the proper environment.

This basically sucks because `bundle exec` is really long to type and the word `exec` is typed by one hand on US keyboards which can slow you down and be error prone.

Ways to get around this are to:

- **Don't use bundle exec** - Chances are things will still work just running the original command but sooner or later you will run into an issue that is related to not running in the bundler environment and finding that out can take a long time.
- **Alias bundle exec** - If you know your way around the shell you can simply alias the bundle exec command to something easier to type like `be`. This can cause problems however if you move between computers a lot.
- **Install Binstubs** - This is my suggested route as it will install "wrapper" commands that will make sure bundler loads everything it needs beforehand. These are then stored in the `bin` directory in your project. So, for example, instead of running:

```
    bundle exec rails server
```

You can now just run:

```
./bin/rails server
```

5.7 How Do I Kill The Rails Server?

Switch back to the terminal where you started `bundle exec rails server` and hit CTRL+c. That's the control key at the same time with the letter `c`. That should terminate the server that is running.

5.8 Wrap Up

In this chapter we covered the process of installing Rails, creating a new project, and then running that project as a smoke test to make sure everything works. If you feel like this is all over your head, don't worry, the remainder of this book will delve into understanding what we just walked through as well as how to find a good development workflow with the tools we have implemented so far.

Chapter 6

Meet The Application (And My Opinions)

Finding an application to develop as a beginner can be difficult. Something too hard or complex can kill the learning process and be tedious for the new developer. Something too simple may not provide enough of a challenge to properly learn the framework. Picking something that is both challenging and approachable is rough.[1]

The application we will create during the course of this book is a personal finance manager which we will call **Bankroll** [2]. Hopefully we will be building something that you can use personally [3] and that is complex enough to give the student a well rounded overview of the framework. The personal finance app will meet the following requirements:

Multiple Users: Users of the application will have a unique account with which to login.

Accounts: Each user can have multiple accounts and users can share accounts.

Transactions: Each account has transactions that record a balance being credited or debited to the account.

[1]Thankfully, that is my job and not yours.

[2]A bankroll is apparently a roll of paper money. I've never had one myself. The term is mainly used by pimps and serial entrepreneurs.

[3]Because tracking your finances is fun! /sarcasm

Permissions: These will allow a user to restrict what other users may do with their accounts

Receipt Tracking: A user will be able to attach a receipt to a transaction for good record keeping.

Email Notifications: Email can be sent when something is triggered like a new user registration or when an account drops below a certain threshold balance.

API: An application programming interface (API) will be created to allow third party systems to retrieve data from the application. If you don't know what an API is, don't worry, it will be covered.

6.1 The Development Cycle (Life Is Not Linear)

It's common for tutorials and books to cover a topic by giving small projects to do or perhaps a larger project but only working forward in a linear fashion. While this is great and I've learned many things these ways, I want to try a different approach that I feel better reflects how things work in "the real world". [4]

The real world is about trial and error. Development that requires you go back and revisit old code, fix those things you never quite got around to, and more importantly, applying what you know now to what you did then.

A good programmer will pretty much hate the code they wrote just a few short months ago. Why? If you're doing it right, you'll be growing both in skill and wisdom. You'll learn new or better ways of doing something. You may find that you're using the right tools the wrong way. Either way, you're eventually going to have to go back and refactor [5] old code. It's a simple fact of life.

As this book progresses we are going to be doing some things in what is considered "The Wrong Way" [6]. The reason is simple: there is a lot to learn and we have to get started somewhere.

[4]Not your parents basement where you hang out with that friend from highschool who spends his days posting enthralling and intelligent rage comics on reddit.

[5]Refactoring means going back to make amends for old sins. Or reworking old code.

[6]Like wearing a brown belt with black shoes. Or wearing those hipster skinny jeans.

So, don't be surprised if we revisit some of our previous code and rework it. Or delete it. Or whatever. Things change. Nothing is written in stone (certainly not software). Change is a fact of life and I want this book to reflect that.

Chapter 7

Digging Around

Now is the time to jump into the app and start messing with things. What we are going to do for the next few chapters is create some basic functionality in our *bankroll* app and then study how this simple Rails application works and where things go.

It may feel like were going to be stuck here for awhile, but as with any complex app it is essential to build our foundational knowledge before moving forward with more code.

7.1 Understanding Scope

Programmers will often refer to all of the things that need to be done for an application to be considered finished as “scope”. The scope of an application is the starting place for all projects. It is from this point you will consider what the application does, how it will work, and what things are critical to implement and what things are not.

I’ve listed the basic scope of our application in the previous chapter so I won’t review it here but we will, however, take the simplest part of the application and start there for our initial development.

The most basic essential to the *bankroll* app is tracking finances. If you really dilute it down to the most critical item it would most likely be the transactions feature. The most essential part of this application is to simply record that something was credited or debited.

So, now we look at this most simplified part of our application and we need to determine what a transaction is so we can go about our merry way implmenting it.

A transaction probably has the following attributes:

- **Date** - The date the transaction occurred.
- **Memo** - What this transaction was for.
- **Amount** - An amount that is either positive or negative to be added/removed from the total

That is probably about as simple as we can get without losing the idea of what a transaction is. Notice here that we are not concerning ourselves with users, accounts, categories, or anything else that for this project would make it actually useful. We are starting from a basic core and working our way up to complexity.

7.2 Using The Rails Generators

Rails comes with commands to help you quickly create objects in Rails. We'll be using the scaffolding generator which will allow us to create and entire feature with one simple command.

To use create scaffolding for our transaction feature simply type:

```
./bin/rails generate scaffold Transaction date:date memo:string amount:decimal
```

This will create a bunch of files for us and will look similar to this:

```
invoke  active_record
create    db/migrate/20120216223753_create_transactions.rb
create    app/models/transaction.rb
invoke    test_unit
create      test/unit/transaction_test.rb
create      test/fixtures/transactions.yml
route  resources :transactions
invoke  scaffold_controller
```

```
    create    app/controllers/transactions_controller.rb
    invoke    erb
    create      app/views/transactions
    create      app/views/transactions/index.html.erb
    create      app/views/transactions/edit.html.erb
    create      app/views/transactions/show.html.erb
    create      app/views/transactions/new.html.erb
    create      app/views/transactions/_form.html.erb
    invoke    test_unit
    create      test/functional/transactions_controller_test.rb
    invoke    helper
    create      app/helpers/transactions_helper.rb
    invoke      test_unit
    create        test/unit/helpers/transactions_helper_test.rb
    invoke  assets
    invoke    coffee
    create      app/assets/javascripts/transactions.js.coffee
    invoke    scss
    create      app/assets/stylesheets/transactions.css.scss
    invoke  scss
    create    app/assets/stylesheets/scaffolds.css.scss
```

Now that we have the basic scaffolding installed for managing transactions we can setup the development database like so:

```
    ./bin/rake db:migrate
```

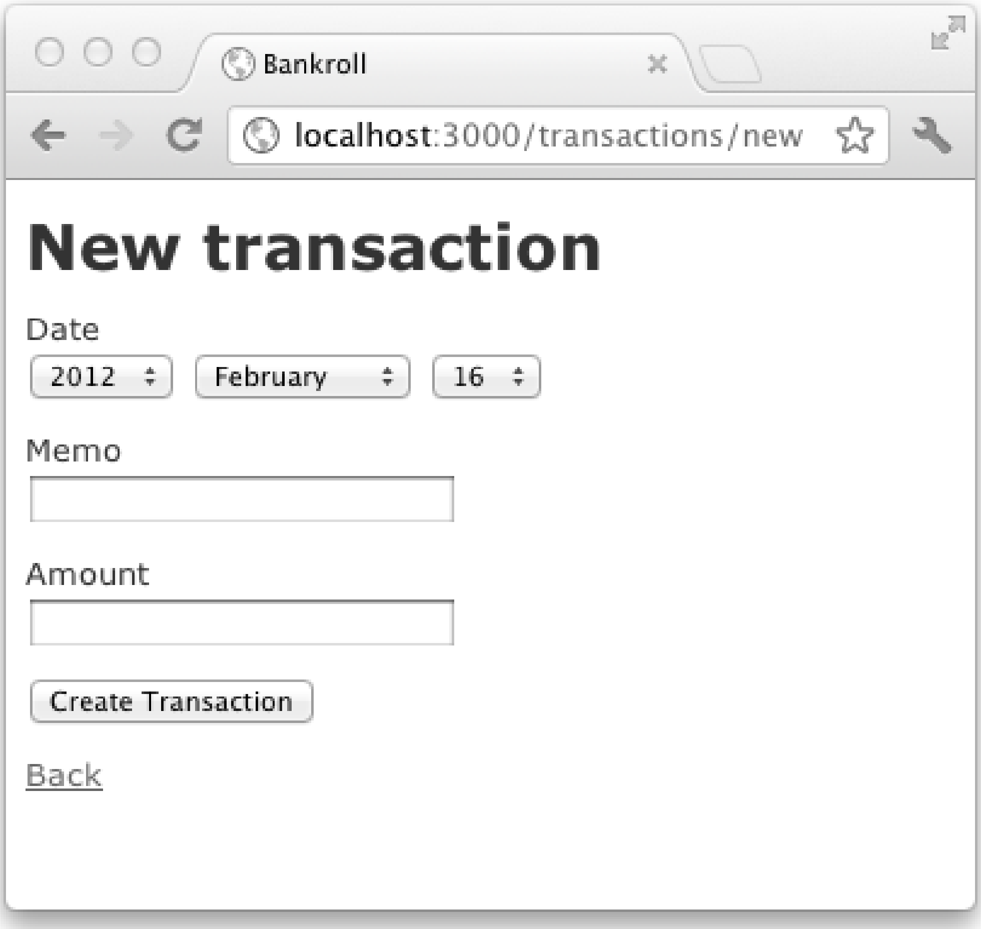
You should see the following output:

```
   ==  CreateTransactions: migrating =============================================
    -- create_table(:transactions)
       -> 0.0011s
   ==  CreateTransactions: migrated (0.0012s) ====================================
```

Now, if we start our the rails development server:

```
    ./bin/rails server
```

You should now be able to go to the `http://localhost:3000/transactions` page and start creating your very own transactions. There are currently no transactions so the page is a little sparse. Clicking on the 'New Transaction' link will take you to the following page in Fig 7-1 where you can fill in the information and create a new transaction.



New transaction

Date
2012 February 16

Memo

Amount

Create Transaction

Back

Figure 7.1: New Transaction Form

7.3 The Black Magic Of Scaffolding

How did we go from a blank application to something that allows us to create, edit, update, and delete a feature? Rails is considered "opinionated software" and as such makes certain assumptions about where things should go and how they should work by default. This makes is easy for Rails to leverage the creation of entire swaths of code without much hassle.

Will we be creating the entire app this way? Absolutely not. I would only recommend creating things this way if your throwing together something quick and dirty that you don't plan on keeping around [1]

The reason we are using it now is that it allows us to get something running so we can then walk through the Rails project and see how things work.

I will clarify here though that the scaffolding is simply one generator Rails provides out of several. Many of the generators are very simple and their use is nice simply because they are quick and do something very menial. The problem with using scaffolding all the time is that it does too much for you that you should be invovled with when creating.

[1]Which, almost always ends up sticking around and becoming important down the road so just do it right the first time.