Practical Ruby Gems

David Berube

Apress®

Practical Ruby Gems

ISBN-13 (pbk): 978-1-59059-811-5

ISBN-10 (pbk): 1-59059-811-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jason Gilmore
Technical Reviewer: Yan Pritzker
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Jeff Pepper, Paul Sarknas, Dominic Shakeshaft, Jim Sumser, Matt Wade
Project Manager: Richard Dal Porto
Copy Edit Manager: Nicole Flores
Copy Editor: Candace English
Assistant Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Diana Van Winkle, Van Winkle Design
Proofreader: Liz Welch
Indexer: Julie Grady
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code/Download section.

Dedicated to my parents

Contents at a Glance

Contents

PART 2 Using Particular Gems

About the Author

DAVID BERUBE is a Ruby developer, trainer, author, and speaker. He's used both Ruby and `Ruby on Rails` since 2003, when he became a Ruby advocate after he wrote about the language for *Dr Dobb's Journal*. Prior to that he worked professionally with PHP, Perl, C++, and Visual Basic.

David's professional accomplishments include creating the `Ruby on Rails` engine for CoolRuby.com (`http://coolruby.com`), a site that tracks the latest Ruby developments, and working with thoughtbot (`www.thoughtbot.com`) on the `Rails` engine that powers Sermo's America's Top Doc contest. He also worked with the Casting Frontier on the `Ruby on Rails` backend that is powering their digital casting services for Los Angeles. He has worked on several other Ruby projects, including the engine powering CyberKnowHow's BirdFluBreakingNews search engine.

David's writing has been in print in over 65 countries, in magazines such as *Linux Magazine, Dr Dobb's Journal*, and *International PHP Magazine*. He's also taught college courses and spoken publicly on topics such as "MySQL and You" and "Making Money with Open Source Software."

Feel free to contact the author via his website at `http://berubeconsulting.com` or via his email address at `djberube@berubeconsulting.com`.

Acknowledgments

I'd like to thank my parents and my sisters; I can't imagine writing this book without them. I'd also like to thank the many friends that have supported me; in particular, I'd like to thank Wayne Hammar and Matthew Gifford.

I'd also like to thank the vast array of professional associates I've worked with and learned from, and in particular I'd like to thank Terry Simkin, Ted Roche, Bill Sconce, Bruce Dawson, K.C. Singh, and Joey Rubenstein. Thanks to Peter Cooper for introducing me to the possibility of writing this book.

Finally, I'd like to thank my editors, originally Keir Thomas and later Jason Gilmore, as well as my technical reviewer Yan Pritzker, my project manager Richard Dal Porto, and my copy editor Candace English.

PART 1

Using RubyGems

This section of the book introduces RubyGems and explains how you can start using them in your code.

CHAPTER 1

What Is RubyGems?

In short, RubyGems lets you distribute and install Ruby code wherever you can install Ruby.

Specifically, RubyGems is a package-management system for Ruby applications and libraries. It lets you install Ruby code—called *gems*—to any computer running Ruby. It can resolve dependences for you, so if you want to install a given piece of software, RubyGems can handle that for you. It can even resolve version dependencies—so that if a certain gem or your code requires a certain version of another gem, it can take care of that. It also wraps all of this functionality in a very easy-to-use package.

The gems come in a variety of types. For example, if you had a Web application to which your users uploaded pictures from a digital camera, you'd likely need to resize the pictures, which come in a variety of sizes. You could write the resizing code by hand, but it'd be considerably faster to use the `rmagick` gem to resize the pictures—and you could add additional features like cropping, rotation, sharpening, and so on with just a few extra lines of code, since `rmagick` includes all of those features. (See Chapter 25 for more details.)

Alternatively, if you want to develop a Web application using `Ruby on Rails`—which is a full-featured, very powerful Model View Controller (MVC) Web framework—you could install that using RubyGems as well. `Rails` consists of a number of libraries and utilities—all of which can be installed by RubyGems with just one command. (See Chapter 23 for more information.)

This chapter covers the features of RubyGems and how it differs from other package-management systems.

Why Use RubyGems?

First of all, RubyGems makes it easy use to install Ruby software. For example, Instiki (`http://instiki.org/`) is a wiki—a kind of content-management system—and if we wanted to install the `instiki` gem, we could do so with the following command in the Linux/Mac OS X shell or the Windows command prompt:

```
gem install instiki
```

Of course, to do that you'd need RubyGems installed, and we'll cover that in the next two chapters. For now, though, you can see how easy it is to install gems—just one command and RubyGems takes care of the rest.

This can be extremely important; for example, if you had a Web application written in Ruby and your server failed, you'd need to be able to quickly and easily install all of the software that your application needs on a new server.

It Provides a Standard Way to Describe Ruby Software and Requirements.

RubyGems lets you define *gemspecs*. A gemspec describes software—it includes the name, version, description, and so forth. This gemspec can be built into a `.gem` file, which is a compressed archive containing the gemspec and all of the files that the software requires.

This `.gem` file can be uploaded to RubyForge, which lets you install it from any Internet-connected RubyGems installation, or it can be distributed via traditional means, like HTTP or FTP. Because the `.gem` file contains a description of the program, you can also use the `gem list` command to see the details of the gem or to search for similar gems. (You can find more details on the `gem list` command in Chapter 3 and you can find out more about building gems in Chapters 34 and 35.)

For example, if you upgrade a version of a gem and the new version has additional requirements, you won't need to scour the documentation for the changes—RubyGems will automatically read the requirements from the gemspec since the format of the gemspecs is standard.

It Provides a Central Repository of Software.

One of the aspects of RubyGems that makes it so appealing is it gives you access to RubyForge—a central repository of Ruby software. You can find out more about RubyForge at, `http://rubyforge.org`.Without RubyForge, you'd have to locate, download, and then install a gem and its dependencies. With RubyForge, though, RubyGems can automatically locate the software and its dependencies for you.

Although most Rubyists (Ruby programmers and enthusiasts) install gems only from the central repository, you aren't required you to use it—you can install gems from any location you choose. (You can also set up your own gem server, which you'll learn about next.) For example, if you had to move your software from one operating system to another, your operating system's packaging system and repository would be different, but RubyGems would stay the same—you can use RubyForge wherever RubyGems is installed.

It Lets You Redistribute Gems Using a Gem Server.

The technology used to serve gems comes with RubyGems. You can set up your own RubyGems server on a local network or on the Internet without much trouble; if, for example, you'd like to cache all of the gems your development team uses on a local server to speed up downloads, you can do that.

If you'd prefer not to use RubyForge and rather distribute gems via your own website or gem server, you can do that too. You can find more details in Chapter 35.

It Handles Software Dependencies for You.

RubyGems can take care of dependencies automatically. That means that when you install a gem, it can automatically determine what other gems are required and ask you if you'd like to install them.

This can make your life much easier, since a significant amount of Ruby software is built using other Ruby software—and that other Ruby software might require still more software. Without RubyGems, you might have to spend hours installing and researching dozens of packages to get complex software working. With RubyGems, you can just install the gem and let it resolve dependencies for you.

It Handles Multiple Software Versions Intelligently.

RubyGems can store multiple gem versions, and software that uses RubyGems can request particular gem software versions—so, for example, an application that requests the `ActiveRecord` gem (`http://rubyforge.org/projects/activerecord/`) could request a gem that's newer or older than a given version. This is very helpful if, for example, a later version of a gem breaks your program, or if your program requires a feature from the latest version of a gem. (You can find more details on how to do this in Chapter 4.)

It Can Be Used Transparently in Place of Regular Ruby Libraries.

RubyGems has a facility that makes it transparent to use gem software. For example, suppose you wanted to use the Camping (`http://rubyforge.org/projects/camping/`) Web microframework. If you installed Camping the traditional way, you would use the Camping library in your code like this:

```
require 'camping'
```

If you installed it via RubyGems, you use the exact same code. As you can see, using code via RubyGems is transparent, so you can switch back and forth easily; if you distribute your software, the user does not need to have RubyGems installed—only the required library.

Note, however, that if you want to require that a certain version of the software is installed, you'll need to use a special RubyGems statement in your code. (See Chapter 4 for further details.)

It Lets You Use the Same Technology on Any Operating System.

RubyGems targets all platforms that run Ruby. If it runs Ruby, it runs RubyGems. A number of other systems exist to make software installation easier; there's everything from those that simply install software—like Window's MSI installation system—to full package-management systems, like Debian Linux's apt (Advanced Package Tool), Red Hat's yum, or OS X's DarwinPorts. Such systems are generally operating system–dependent, though, as we'll discuss next.

How Does RubyGems Compare to Other Packaging Systems?

Operating system–specific packaging systems, such as apt or yum, can carry Ruby software as well. Since Ruby software can be used by non-Rubyists, this is important. It's also convenient if you need just a few pieces of software. For Rubyists, though, it's usually better to install gems using RubyGems, since RubyGems has the best selection of Ruby software and the latest versions. Additionally, unlike RubyGems, OS-native packagers don't handle multiple gem versions installed simultaneously.

However, it is possible to install a limited selection of Ruby software using other packaging systems. For example, you could install the MySQL Ruby bindings via gem like this:

```
gem install mysql
```

Alternatively, you could install the same library via apt-get under Ubuntu Linux like this:

```
apt-get install libmysql-ruby1.8
```

Finally, you could install it via DarwinPorts under OS X like this:

```
port install rb-mysql
```

Note that those three commands require you to be logged in as `root`—if you prefer, you can prefix each command with `sudo`, which will execute that single command with the root-user privileges.

In some cases you can install software via apt or another packaging system. Such systems usually have a very limited selection of Ruby packages, but if they happen to include all of the software you need, you may be able to use them. Consult the documentation that comes with your Linux distribution or other packaging system.

Note, though, that installing gems from your OS distribution is not recommended. It means you have to use the version of the software in your OS's repository, and often this lags significantly behind the RubyGems versions. Using the gem installer, as we do throughout this book, will automatically give you access to the most recent gem versions.

Of course, you can always skip package-management systems entirely—you can install Ruby software by running an install script manually or by copying files into the `lib` directory of your Ruby installation. If you want to do so, download the software you want from its homepage and consult the included `README` or `INSTALL` file.

CHAPTER 2

Installing RubyGems

You'll need the RubyGems system to follow the examples from this book. The RubyGems system lets you use a vast array of Ruby software packages—including all of the gems we cover in this book. In general, it's fairly easy to install RubyGems. Of course, before you can install RubyGems, you need to install Ruby—we'll cover both in this chapter. Finally, we'll explain how you can update a RubyGems system you've already installed.

Note If you already have RubyGems installed, you can skip this chapter.

Installing Ruby

To follow the examples in this book, and before you install RubyGems, you must have the Ruby programming language interpreter and libraries installed on your machine.

Mac OS X comes with Ruby preinstalled; if you have Mac OS X installed, you can skip straight to the section, "Installing RubyGems under Linux and Mac OS X." Many Linux distributions include Ruby, so we'll cover how you can check if your computer has Ruby installed. Windows does not install Ruby by default, so if you are running Windows, skip straight to "Installing Ruby on Windows Using the One-Click Installer." (If you're using Windows, the One-Click Installer will install both Ruby and RubyGems at once.)

Is Ruby Already Installed on Your Computer?

If you're not sure if you need to install Ruby, run the following command at the OS X/Linux shell or the Windows command prompt:

```
ruby -v
```

If you receive a "command not found" error, you don't have Ruby installed. You should get a message like the following:

```
ruby 1.8.4 (2006-04-14) [i386-mswin32]
```

The version number is the number immediately after the "ruby"—in this case, 1.8.4. Note that to use the RubyGems system (as well as to use a lot of other software that uses Ruby), you'll need version 1.8.4 or later. If you have a lower version, you should upgrade—look for appropriate instructions for your operating system to upgrade your installation.

Installing Ruby on a Linux System

We'll briefly cover three methods of installing Ruby on Linux. The first, apt, is a package manager for Debian-based distributions, such as Ubuntu. The second, yum, is a package manager for Red Hat–based distributions. These both offer an easy way to install Ruby. If your system does not support either apt or yum, you can install Ruby by compiling the source code yourself, which is slightly more complicated. We'll cover that method last.

Installing Ruby on Debian Linux Distributions with apt

apt is a popular package-management system for Debian Linux and Debian-based distributions, such as Ubuntu Linux, Lindows, Xandros, and others. It bears some similarities to RubyGems; for instance, it can download, install, and remove software from the command line. (apt can also be used on non-Debian distributions, but it does not come installed by default.)

If you'd like to use apt to install Ruby, you can do so as follows:

```
sudo apt-get install ruby*
```

This will automatically download and install Ruby, and you can proceed to the "Testing Your Ruby Installation" section of this chapter.

■ **Note** The last two `apt-get` commands install required libraries for installing RubyGems; if you don't plan on installing RubyGems, those aren't absolutely necessary to run Ruby.

Installing Ruby on Red Hat Linux Distributions with yum

yum is another package-management system—it's very similar to apt. It's available on all versions of Fedora Core. If you'd like to use yum to install Ruby, you can do so as follows:

```
yum install ruby
```

This will download and install Ruby and the required libraries for you, and you can proceed to the "Testing Your Ruby Installation" section of this chapter.

Installing Ruby on Linux from the Ruby Source

You'll need to have `gcc` and `make` installed to compile Ruby; if you don't have them installed, consult your distribution's documentation for the installation instructions. First download and uncompress the latest Ruby source tarball from `ftp://ftp.ruby-lang.org/pub/ruby/`, then compile and install Ruby with the following shell commands:

```
./configure
make
make test
make install
```

Once you've done so, your Ruby installation should be ready and you can proceed to the "Testing Your Ruby Installation" section of this chapter.

Installing Ruby on Windows Using the One-Click Installer

The Ruby One-Click installer is very easy to use. It is a precompiled, self-contained Windows installer. It's developed by Ruby Central (`http://rubycentral.org/`), and provides Ruby in the only real way to distribute software for Windows, which is as a binary—after all, Windows does not have any method to compile software by default. (This is true under some Linux distributions as well.)

The installer is very simple, as you can imagine from the name—you won't have to launch multiple programs or type commands into the Windows prompt, so it fits in well with the Windows way of doing things.

You can download the One-Click Installer from `http://rubyinstaller.rubyforge.org/`. Once you've done so, run the program by double-clicking on the icon. You'll be asked a few questions, but if you select the default options you should be all set. Proceed to the "Testing Your Ruby Installation" section of this chapter.

DOWNLOADING FILES WITH WGET

While the most familiar way to download files under Windows is to use a Web browser, there are other options. A popular Linux utility, wget, comes in Win32 form—you can get it at `http://users.ugent.be/~bpuype/wget/`.

wget lets you download files from the command line in just one command. Not all Windows users are comfortable using the command prompt, but once you become comfortable, many people find it easier to use.

Once you've downloaded and installed wget, you can use it to download Ruby from the command prompt like this:

```
wget http://rubyforge.org/frs/download.php/11926/ruby184-20.exe
```

This would save you a number of clicks and the hassle of launching a Web browser.

Testing Your Ruby Installation

How can you be sure that Ruby works? Let's try a very simple test. You can rerun the version command discussed earlier by typing the following command in the Windows command prompt or the Linux/OS X shell:

```
ruby -v
```

You should get a display similar to the following:

```
ruby 1.8.4 (2006-04-14) [i386-mswin32]
```

If you'd like to test an actual line of code, you can do so as follows:

```
ruby -e "puts 'hello world!'"
```

You should get the following output:

```
hello world!
```

Once you've verified that you have Ruby installed, you can install RubyGems; we'll cover that next. If you used the Windows One-Click Installer, it already installed RubyGems for you, so you can skip straight to "Testing Your RubyGems Installation."

Installing RubyGems Under Linux and Mac OS X

It's fairly easy to install RubyGems on Linux, and you can use the same procedure to install RubyGems on Mac OS X. On those systems, use the following shell commands to download and install RubyGems:

```
curl -O http://rubyforge.org/frs/download.php/11289/rubygems-0.9.0.tgz
tar -xvzf  rubygems-0.9.0.tgz
cd rubygems-0.9.0.tgz
ruby setup.rb
```

Once you've done so, set an environment variable, `RUBYOPT`, using a line in your `.profile`:

```
export RUBYOPT=rubygems
```

This environment variable causes RubyGems to be run whenever Ruby is run. You can get an similar effect by including the line `require "rubygems"` in all of your Ruby scripts, but since most Ruby scripts using RubyGems are written assuming that you have `RUBYOPT` set, that'll require you to modify all of the programs you download—no small task. You can also run your Ruby script with the `-rubygems` option, but that's a lot of extra typing.

■ **Note** You shouldn't have any negative effects from setting the `RUBYOPT` variable—even if some of your scripts don't use RubyGems. (Keep in mind that the Windows installer sets the `RUBYOPT` variable for you unless you explicitly tell it not to.)

At this point, you should have a working RubyGems system, so you can install and use gems. You can now check your installation by following the directions in the next section of this chapter.

Testing Your RubyGems Installation

First let's pull up a list of installed gems. You can use the following command at the Linux/OS X shell or the Windows command prompt:

```
gem --version
```

You should get the following response:

```
x.y.z
```

Note that `x.y.z` will be replaced by the appropriate directions for your operating system. If you get a "command not found" error, you've done something wrong and you'll want to follow the appropriate instructions again for your operating system; you'll also want to make sure you installed Ruby before you installed RubyGems. Also check that if you already had Ruby installed, it's a version later than 1.8.4. If not, you'll want to install a more recent version.

At this point, you have a working RubyGems install—you can now try all of the examples in this book.

Updating Your RubyGems System After You've Installed It

Once you've installed RubyGems, you can update it easily. You can use the same command on any operating system. Typing the following command at the Linux/OS X shell or the Windows command prompt will update RubyGems:

```
gem update --system
```

This will automatically download and install the latest update of the RubyGems system; you can then use whatever updates have been made available. To check for RubyGems updates, visit `http://rubygems.org/`.

CHAPTER 3

Using RubyGems in Your Code

In this chapter you'll learn how you can use RubyGems in your code. You'll learn about installing individual gems; see a practical example of using them; get debugging tips; and consider a few miscellaneous issues, like unpacking gems so they can be edited, freezing gems so they don't change, and using plugins and engines under `Rails`.

Getting Started with a Ruby Gem

Before we use a gem, we must install it. The instructions in this chapter assume you have the RubyGems system already installed; if you don't, refer to the previous chapter.

Gems are usually downloaded automatically, since the gem program can fetch gems from the Internet. In fact, you can install most gems with a single command:

```
gem install gemname
```

Replace *gemname* with the name of the gem you want to install. Of course, to do that, you need to know what the gem is called. To demonstrate this and other aspects of gem use, we're going to follow a small demo project.

Suppose you are developing an ecommerce application, and you want a quick way to find out if a credit card number is valid without actually charging the card; that way, you can have immediate feedback in your user interface if a customer mistypes the number.

To find a gem that fits our criteria, we could do a Web search to determine if there are any gems with the functionality we need. However, we can first search the gem repository directly using the `gem list` command. We can guess that a gem dealing with credit will start with the word *credit*; let's search the repository and see what we get. We can do that with the following shell or Windows Prompt command:

```
gem list -r credit

*** REMOTE GEMS ***

creditcard (1.0)
 These functions tell you whether a credit card number is
 self-consistent using known algorithms for credit card numbers.
```

A few things to note: the `-r` switch tells the `gem` command not to search the local repositories, since we'd probably know if we installed a gem that fits our needs. If you omit the `-r` switch, it'll search both local and remote gems. If you replace `-r` with the `-l` switch, you'll search local gems only.

The `credit` part of that command tells the `gem` command to search for gems whose name starts with `credit`. Note that this is part of a regular expression, so if you say `gem list -r .*credit`, you'll search for any gem whose name contains `credit` anywhere in the string.

Also note that you do not need to specify a search criteria; `gem list -r` will give you a complete list of remote gems, and `gem list -l` will tell you all of the gems you've installed locally. (You can save a copy of the remote gem list using your operating system's redirection support: `gem list -r > remote_gem_list.txt` will save a list of all remote gems available into `remote_gem_list.txt`.)

Now that we know the name of the gem, we can install it. Here's the command that will install the `creditcard` gem:

```
>gem install creditcard
Successfully installed creditcard-1.0.0
```

You'll likely need to run this command as root under Linux/OS X. Once you've installed the gem, you can create an application to use it.

Using the creditcard Gem

The `creditcard` gem verifies that credit card numbers are valid. At first glance, it might seem like the gem actually runs cards through a credit card processor; it doesn't. It also does not verify that the account exists, that the expiration date is correct, or that there is sufficient available balance in the account to make a charge; all of those require actually charging the card via a payment gateway or merchant account, which takes time and isn't done until an order is complete.

■ **Note** Keep in mind that any given gem won't always solve your problem—you might need to look around a bit to find one that fits, and even when you find it you'll likely need to do some work to get it to solve your particular problem. At times, it may be more work fitting the gem into place than it would be to solve the problem from scratch, particularly if the gem were badly designed—in that case, you'd be better off using custom code.

However, it does verify that a number isn't invalid; it checks the internal checksum of the card number, and that can be done immediately as a user is entering card information. As a result, the `creditcard` gem can help ensure that users entered cards correctly and do not make any typos. Let's write a simple app to use the `creditcard` gem to test credit card numbers.

```
require 'creditcard'

if ARGV[0]
  credit_card_number=ARGV[0]
  if credit_card_number.creditcard?
    puts "Credit card number is valid " <<
    "with type #{credit_card_number.creditcard_type}."
  else
```

```
    puts "Credit card number is not valid."
  end
else
  puts "Please enter a valid credit card number."
end
```

That's pretty simple code for some fairly complex functionality; the statement "require creditcard" gives us the ability to use the full functionality of that gem quite easily on any string.

■ **Note** You can download all of the code from this book from the Source Code/Download section of `http://www.apress.com/` instead of typing it in.

The `ARGV` array is a Ruby global variable that contains the command-line arguments to the program. Our program expects you to pass a credit card on the command line, so if there aren't any, the program will print "Please enter a valid credit card number." It'll then call the `creditcard?` method. This returns true if the string is a valid credit card, and false otherwise. This method takes an optional parameter—if we called it `credit_card_number.creditcard? visa`, it would return true only if the number were a valid Visa credit card number, and false if it were an invalid credit card number or a non-Visa credit card number. The other method we use is the `creditcard_type` method; it's also an extension to the `String` class. That method returns the credit card type, and the preceding listing uses it to print out the credit card type.

Note that no special `creditcard` objects are created; the `creditcard` gem extends the `String` class directly. This is not possible in most languages; however, in Ruby this is called *monkeypatching*, and is common. Also note that both methods end in a question mark. This is a Ruby convention indicating that the method returns a true or false value. The question mark has no special syntactic value—it's just an indication to the programmer. (The other symbol commonly used at the end of method names is the exclamation mark, which means that a method modifies the receiver in place.)

Let's test the program. We'll start by checking that a completely bogus input doesn't work:

```
ruby creditcard_check.rb not-a-number
```

```
Credit card number is not valid.
```

It's good so far. Now let's try with a correctly formatted number that isn't a valid card:

```
ruby creditcard_check.rb 0000-0000-0000-0000
```

```
Credit card number is valid with type unknown.
```

The numbers in Table 3-1 are test card numbers used to debug payment gateways, terminals, and merchant accounts. They are numerically correct, but aren't attached to any charge account. We can use them to test our script.

Table 3-1. *Test Credit Card Numbers*

Card Type	Test Number
Visa	4111-1111-1111-1111
MasterCard	5431-1111-1111-1111
American Express	341-1111-1111-1111
Discover	6011-6011-6011-6611
Diners Club	3530-1113-3330-0000

Let's grab the test numbers from the table and see how well they work:

```
ruby creditcard_check.rb 4111-1111-1111-1111

Credit card number is valid with type visa.

ruby creditcard_check.rb 5431-1111-1111-1111

Credit card number is valid with type mastercard.

ruby creditcard_check.rb 341-1111-1111-1111

Credit card number is valid with type american_express.

ruby creditcard_check.rb 6011-6011-6011-6611

Credit card number is valid with type discover.

>ruby creditcard_check.rb 35301113333300000

Credit card number is invalid.
```

You can see that the gem detects valid test cards without a problem. In a production environment it may be wise to test with a few real cards as well. You can also see that it supports both card numbers formatted with dashes and those without, and a fair number of card types.