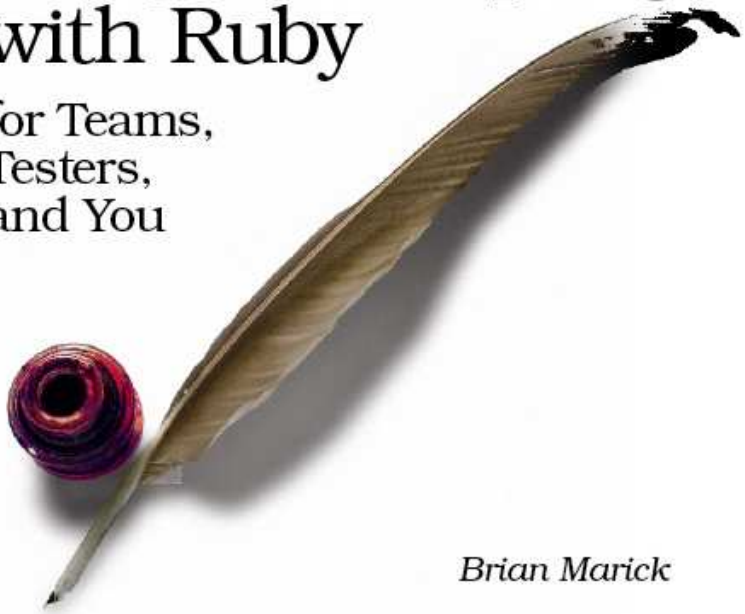
The Pragmatic Programmers

Everyday Scripting with Ruby

for Teams, Testers, and You



Brian Marick

What readers are saying about
Everyday Scripting with Ruby

What a wondrous collection of recipes, guidelines, warnings, comprehensive examples, metaphors, exercises, and questions! It's a terrific value to software testing practitioners who want to get the most from their test automation effort.

► **Grigori Melnik**
Lecturer, University of Calgary

A fantastic type-along-with-me introduction to a powerful scripting language that starts in the shallows and then moves into the depths turning the reader into an accomplished Ruby scripter, almost without them noticing it!

► **Erik Petersen**
Emprove

Finally a hands-on book that is filled with gems of wisdom for the testing community. By following the book's easy-to-read chapters, real-life code samples, and superb coverage of complex topics like test-driven design and inheritance, a tester will not only take her testing career to the next level but also contribute immensely to the software development at her organization.

► **Gunjan Doshi**
VP of Product Development and Process Excellence,
Community Connect, Inc

Marick explains the Ruby language using a series of short, practical examples. Watir users and other testers who want to learn Ruby will find it very accessible.

► **Bret Pettichord**
Lead Developer, Watir

When you've read this book, you will be able to automate software tests, which will give you an edge on most of your QA workmates. You will be able to program in Ruby, which is a joy in itself. You will have created several very useful utilities and will know how to adapt them to meet your particular needs. All of the above will have been achieved briskly and pleasantly. You will become a more effective tester and, most likely, will have a fine time in the process.

► **George Hawthorne**
Consultant, Oblomov Consulting

The book is an excellent read, is very informative, and covers a lot of ground in a relatively slim book. I think this is always a good idea. I have a lot of 800+ page tech books that I've read about the first half or two thirds of, because they are padded toward the end with very esoteric information. This book held my interest throughout—I have a full-time job and a ten-month-old son and still managed to get through these examples in around a week! Brian's personality comes through (e.g., the Kennel containing Barkers) in a good way that helps rather than hinders in understanding the material.

► **Paddy Healey**
Enterprise Systems Engineer, Aventail Corporation

The chapters, examples, and exercises on regular expressions are worth the cost of the book alone! Everything else is more than just gravy—it's every kind of dessert you didn't know you could have. Whether you are just beginning to script or have been scripting for several years, this book will be an invaluable resource. The examples and exercises, Ruby facts, step-by-step approach, and explanations will help you kick up your automation efforts to a whole new level!

► **Paul Carvalho**
Consultant, Software Testing and Quality Services

Everyday Scripting with Ruby

For Teams, Testers, and You

Brian Marick

The Pragmatic Bookshelf
Raleigh, North Carolina Dallas, Texas

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at

http://www.pragmaticprogrammer.com

Printed in the United States of America.

ISBN-10: 0-9776166-1-4

ISBN-13: 978-0-9776166-1-9

Printed on acid-free paper with 85% recycled, 30% post-consumer content.

First printing, December 2006

Version: 2007-1-25

To Dawn, my Best Beloved, best friend, and role model
And to shoemakers' children everywhere

Contents

Chapter 1

Introduction

The shoemaker's children are running around barefoot.

People on the outside of software development projects see them spew out a multitude of tools that shift work from people to computers. But the view inside a project is—all too often—different. There, we see days filled with repetitive manual chores. At one desk, a tester is entering test data into a database by hand. At another, a programmer is sifting through the output from a version control system, trying to find the file she wants. At a third, a business analyst is copying data from a report into a spreadsheet.

Why are these people doing work that computers could do perfectly well? It's a matter of *knowledge* and *skill*. The tester thinks programming is too hard, so he never learned. The programmer knows programming, but none of her languages makes automating this kind of job easy, and she doesn't have time to do it the hard way. The analyst once wrote a script to do a similar chore, but it broke when she tried to adapt it to this report. Getting it working would take more time than copying the data by hand, even if she has to copy it six times over the next month.

Joe Asks. . .

Scripting? Programming? What's the difference?

There isn't one. I'm using "scripting" for this book because it sounds less imposing and more suited to everyday chores.

This book is for all those people.

- *For the person who thinks programming is too hard* (our tester): it's not as hard as all that. Programming has a bad reputation because computers used to be too slow. To make programs run fast enough, programmers had to use programming languages that made them tell the computer all kinds of fiddly details. Computers are now fast enough that we can use languages that make *them* figure out the fiddly little details. As a result, programming is now much easier.
- *For the person who gets bogged down when writing or changing larger scripts* (our analyst): you don't yet have the skills to master complexity. This book teaches them. It's a tutorial in the modern style of programming, one that emphasizes writing tests first (test-driven programming), borrowing other people's work in bits and pieces, growing programs gradually, and constantly keeping them clean.

  Many scripts will be one-shot: write it, use it, throw it away. But for scripts you plan to keep around, these skills will let you do it. (In truth, many professional programmers I meet haven't yet learned these particular skills, so they will find this book a useful introduction.)
- *For the person who knows the wrong languages well* (our programmer): languages like Java, C#, C++, and C are perfectly fine languages—in their niche. But their niche is not writing smaller programs quickly, especially not smaller programs that manipulate text and files rather than numbers and internal data structures. You need to add another language to your repertoire.

In this book, you'll learn a language—Ruby—that is well suited to each of these three audiences. It's easy to learn and quick to write. While it has the features needed for simple scripts that transform or search text, it also has all the features needed to cope with complexity. If you're a tester, you'll be pleased to know that testing is considered one of Ruby's niches (largely due to Watir, http://wtr.rubyforge.org/, a tool for driving web browsers). If you're a programmer, you may already know that Ruby has recently become explosively popular because of its "killer app," Rails (a framework for building web applications, http://www.rubyonrails.org/). Despite that, it's more than a decade old, so it's not just some passing fad or unstable prototype. And everyone will be pleased with the Ruby community, which is notably friendly.

1.1 How the Book Works

This is a *hands-on* book. Scripting is like riding a bicycle: you don't learn it by reading about it; you learn it by doing it. And you get better by doing more of it. The purpose of a book, or of a coach, is to direct your practice so that you get better faster.

Therefore, the book is organized around four separate projects that are similar to those you might do in real life. I build the first two projects slowly, showing and explaining all my work. You'll learn best if you type along with me, building the project as we go. In the third and fourth projects, I move faster and explain only the finished result.

practice files

The *practice files* that come with the book contain a series of snapshots for each of the first two projects. The snippets of Ruby code in the book identify the file they come from. You can look at the file to see the snippet in context, to diagnose problems by comparing what you've typed to what I have, or to start your own typing in the middle of a project instead of at the beginning.

Some of you won't create the projects along with me. I do still urge you to work through the exercises and compare your solutions to the solutions I give.

The Projects

The first project is an uninstaller checker. If you uninstall your company's product, does the uninstaller remove everything it should? Does it remove something it shouldn't? This script will tell you. More generally, it lets you take snapshots of any part of your hard disk and compare them.

The second project reaches out to a version control system, retrieves change information, and summarizes it for you. It's a typical example of manipulating text.

The third project visits to a website, "scrapes" data out of it, and puts that data into a comma-separated value file for use by a spreadsheet.

The final project is a "watchdog" script. It can watch long-running programs or tests and then send you an instant message or email when they finish.

A Special Note to Testers

You were the original audience for this book. It used to be called *Scripting for Testers*, but people kept saying it would be useful to a broader audience. Even programmers I expected to be uninterested said things like "with only a few changes, this book would be for me." So I made the changes, but testers still have a special place in my heart.

As a tester, I bet you came to this book hoping to learn how to automate test execution: how to push inputs at a program (probably through the user interface), collect the results, and compare what the program produced to what it should have produced. Even when this book was exclusively for testers, I didn't create any projects like that. I had two reasons:

- *Automating test execution is not the most efficient way for you to learn.* I aim to teach you the practices, habits, and Ruby features you'll need in real life. You don't need those things to write one automated test or even ten, maybe not even a hundred, so it would feel artificial, false, and unconvincing for me to teach them in the context of a small automated test suite. They're better taught with small projects of a different sort.
- *Automating test execution may not be the most effective thing for you to do.* Is test execution the *only* task you do by hand? Probably not. People overly focused on test automation often miss opportunities for simple scripts that yield outsized improvements.

1.2 An Outline of the Book

This is a book about both the features of Ruby and the craft of scripting. Each part of the book teaches some of both. Ruby features are introduced as they're needed for that part's project. Each part also introduces new skills that build on earlier ones.

Part I, on page 30, teaches you the basics of Ruby and the basics of scripting. If you've never programmed, work through it carefully. If you already know a language, you can read it more casually, but do still read it. Ruby is based on ideas you might not know and has features you may not have seen before; if you skip them, you won't be prepared for the rest of the book.

At the end of Part I, all three kinds of reader will be ready to learn how to script better. Part II, on page 67, adds more Ruby facts, but it's mainly about teaching you how to write scripts in a steady, controlled way. All programmers know the feeling of hitting that wall where they can't make any change without breaking something. I want to show you how to push that wall further away.

Part III, on page 139, concentrates on accomplishing more with less effort. It shows how to save work by finding, understanding, and including libraries written by others. It shows you how to set up your scripts so that your co-workers can download, install, and use them easily. While demonstrating still more features of Ruby, this part also elaborates on an important topic from Part II, "regular expressions," a powerful way of searching text.

Part IV, on page 213, covers the advanced topic of inheritance. Inheritance can sometimes save even more work than libraries because someone else designs a framework for part of your script. You need only plug in pieces that the framework orchestrates. Part IV shows you both how to use complicated frameworks others create and how to make simpler ones for yourself. You may want to get experience writing scripts of your own before learning about frameworks.

The book ends with a glossary, solutions to exercises, and an index. What else? Throughout the book, you'll find chapters called "Ruby Facts." When I introduce a Ruby feature in the process of creating a script, I'll describe only the bits used in the script we're writing. But you'll want to know more about such features when you write your own scripts, so I use the fact chapters to tell you more. Skip them if you like.

Despite those chapters, this book is not a complete reference on Ruby. Eventually you'll want to buy one. I heartily recommend Dave Thomas and friends' *Programming Ruby* [TH01]. It's also from the Pragmatic Bookshelf—indeed, Dave is one of the owners of the press. But I'm not recommending their book because they're my publisher. They're my publisher because I kept recommending their book.

1.3 Service After the Sale

Everyday Scripting with Ruby has its very own Pragmatic Programmers' web page at http://www.pragmaticprogrammer.com/titles/bmsft/. There, you will find updates, errata, source for all the examples and more.

1.4 Supplements

As time and demand permit, I'll be publishing supplements to this book; each will be devoted to a particular topic. Please check the book's home page for details.

1.5 Acknowledgments

This book would not exist were it not for the prodding of Bret Pettichord.

Thank you, those who commented on drafts: Mark Axel, Tracy Beeson, Michael Bolton, Paul Carvalho, Tom Corbett, Bob Corrick, Lisa Crispin, Paul Czyzewski, Shailesh Dongre, Gunjan Doshi, Danny Faught, Zeljko Filipin, Pierre Garique, George Hawthorne, Paddy Healey, Jonathan Kohl, Bhavna Kumar, Walter Kruse, Jody Lemons, Iouri Makedonov, Chris McMahon, Christopher Meisenzahl, Grigori Melnik, Sunil Menda, Jack Moore, Erik Petersen, Bret Pettichord, Alan Richardson, Paul Rogers, Tony Semana, Kevin Sheehy, Jeff Smathers, Mike Stok, Paul Szymkowiak, Jonathan Towler, and Glenn Vanderburg.

Special thanks to Paul Carvalho for teaching me something I didn't know about Windows and for working through Part IV before Part III, and to Paul Czyzewski for how thoroughly he reviewed the pages I gave him time to review.

My editor, Daniel Steinberg, provided just the right mix of encouragement, support, and pressure.

I'll be eternally grateful to my publishers, Andy Hunt and Dave Thomas, for not seeming to mind as their children were born, grew up, left home, got married, and had children of their own—all during the writing of this book.

And I'd like to thank my family. You wouldn't *believe* what they've let me get away with.

Chapter 2

Getting Started

This chapter gets you ready for the rest of the book.

- Everyone will need to download the practice files.
- If you're not familiar with the *command line* ("the DOS prompt," "the shell"), you'll need to learn a bit about it.
- Ruby might be preinstalled on your system. If it isn't, you'll need to install it.
- Anytime you type, you make typographical errors. Typing scripts is no different. You need to learn to recognize the signs you've made a mistake.

2.1 Download the Practice Files

This book comes with a number of Ruby scripts you can practice on. You can download them as a zip archive from the book's web page at http://www.pragmaticprogrammer.com/titles/bmsft/. Download it anywhere you please.

Your browser might "unzip" the file for you when you download it. If not, double-clicking or right-clicking it will probably work. Failing that, on Mac OS X and other Unix variants you can type unzip bmsft-code.zip to the command-line interpreter.[1] On Windows, download an application like WinZip (http://winzip.com/), and set it to work.

Unzipping the file creates a folder named code. I recommend renaming that to something more specific, like scripting-book, but I'll use code to refer to it throughout the book.

1. The command-line interpreter will be explained shortly.

Within code, there is a subfolder for each of the scripts in the book. Do your work within those subfolders. There's also a subfolder with solutions to the exercises and several subfolders with more Ruby examples.

2.2 In the Beginning Was the Command Line

When you use Ruby or any other scripting language, you're likely to use your computer's *command-line interpreter*. The command-line[2] interpreter is a program that lets you command the compiler by typing in text, rather than by pointing and clicking with a mouse. If you've never used the command-line interpreter, here's an introduction.

command-line interpreter

Windows

In Windows, you get to the command-line interpreter from the Start menu. Click the Run menu item, type cmd, and then press Enter. You'll see something like this:

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:>
```

The C:\> you see is called the *prompt*. It's called that because it's supposed to prompt you to type some commands for the computer to execute. Not everyone who reads this book will have the same prompt, so I've arbitrarily chosen to show the prompt as prompt> from now on, except when I'm talking about something specific to Windows. When you see an instruction to type something like this:

```
prompt> irb
```

I want you to type i r b Enter. Don't type the prompt. Let's suppose you installed the practice files in C:\unzip-place. Type this:

```
C:\> cd c:\unzip-place\code
```

(cd stands for "change directory"—"directory" is a synonym for "folder".)

When you change to a folder, it becomes your *current working folder*. If a command doesn't name a specific folder, the command-line interpreter assumes you mean the current working folder. For example, you can view the contents of the current working folder like this:

current working folder

```
C:\unzip-place\code\> dir
```

2. The title of this section refers to an essay by author Neal Stephenson. You can find it at http://www.cryptonomicon.com/beginning.html.

If you're working on the file-inventory project, you can now go there like this:

```
C:\unzip-place\code\> cd inventory
```

Note that you don't have to preface inventory with C:\unzip-place\code because that's your current working folder.

You can move back up to the enclosing folder like this:

```
C:\unzip-place\code\inventory\> cd ..
```

That's all you need to know to run the examples in this book (though you'll want to learn more).

Mac OS X, Linux, BSD, and Other Unix Variants

The Mac command prompt is an application named Terminal. It lives in the Utilities subfolder of the Applications folder. On other Unix variants,[3] the command prompt might be named Konsole, Terminal, gnome-terminal, or xterm. You should be able to find it in one of your window manager's menus.

Regardless of how you start it, the command prompt looks something like this:

```
Last login: Sat Dec 16 11:45:37 on ttyp1
Welcome to Darwin!
computer-name:~ user$
```

The computer-name:~ user$ you see is called the *prompt*. Your prompt is probably different, so from now on, I'll show the prompt as prompt>. When you see an instruction to type something like this:

```
prompt> irb
```

I want you to type i r b Return. Don't type the prompt.

Let's suppose you installed the practice files in the unzip-place folder in your home folder. Type this:

```
prompt> cd ~/unzip-place/code
```

cd stands for "change directory"—"directory" is a synonym for "folder." The twiddle (~) means your home folder.

3. Mac OS X, the other systems in the title of this section, and still others I didn't name all have a common ancestry: the Unix operating system developed at Bell Labs in the 1970s. To a scripter, Mac OS X is just Unix with an exceptionally pretty face added. Oh, and it has some nice applications too.

When you change to a folder, it becomes your *current working folder*. If a command doesn't name a specific folder, the command-line interpreter assumes you mean the current working folder. For example, you can view the contents of the current working folder like this:

current working folder

```
prompt> ls
```

If you're working on the file-inventory project, you can now go there like this:

```
prompt> cd inventory
```

Note that you don't have to preface inventory with ~/unzip-place/code because that's your current working folder.

You can move back up to the enclosing folder like this:

```
prompt> cd ..
```

That's all you need to know to run the examples in this book (though you'll want to learn more).

2.3 Do You Need to Install Ruby?

Ruby runs on Windows 2000, Windows XP, or later; Mac OS X; and any version of Unix you're likely to find. You may already have Ruby installed on your machine. To find out, type this at the command prompt:

```
prompt> ruby -v
```

If you see a complaint like "command not found," you'll have to install Ruby.

If Ruby is installed, the response will look something like this:

```
ruby 1.8.1 (2003-12-25) [powerpc-darwin]
```

The version of Ruby shown there, 1.8.1, is older than the one I used when writing this book. I used 1.8.2. All the examples here might work perfectly, but I wouldn't count on it. Install the latest version.

2.4 Installing Ruby

Windows

There is a one-click Ruby installer. You can find it here: http://www.ruby-lang.org/en/downloads/. After you download it, double-click it in Windows Explorer to run it, and then follow the directions.

After installing Ruby, close any command-line windows, open a new one, and then follow the directions in Section 2.3, *Do You Need to Install Ruby?*, on the preceding page, to check that it was installed correctly.[4]

Mac OS X

Tiger (version 10.4) and later versions of Mac OS X come with recent enough versions of Ruby. If you're using an older release of OS X, see http://www.ruby-lang.org/en/downloads/ or the book's website for instructions.

Other Unix Variants

You may be able to find precompiled versions of Ruby (RPMs, etc.) in the usual places and retrieve them via the usual tools (apt-get, pkg-get, ports, etc.). Otherwise, see the book's website for instructions.

2.5 Your Two Basic Tools

There are two basic tools: an editor and an interpreter.

Your Editor

You can use any editor that works with text files to create Ruby scripts. On Windows, I recommend you use SciTE, which is installed with Ruby. It's more than just a text editor: it understands Ruby well enough to color-code parts of a script to make it easier to read, and it lets you run scripts without having to switch to the command line. (In the Start menu's Programs entry, you'll find a Ruby entry, and SciTE is under that.)

On a Mac, I recommend TextMate (http://macromates.com/). It costs money, but you can try a free download.

On the Mac and other Unix-like systems, you can use pico. It's free. Start it by typing its name at the command prompt. It shows its available editing commands at the bottom of the screen. In that help, Control+X is denoted by ^X.

If you use the Gnome window system on Linux, gedit is worth trying.

4. You have to close and open a new command line because the old one may not "notice" that Ruby has been installed.

irb

The second useful tool is irb. It lets you try your ideas without having to write a whole script. You can type a little snippet of Ruby and quickly check what it does. You'll see many examples later in the book. For now, check that irb is ready for use. At the command prompt, type the following. (Remember not to include the prompt.)

```
prompt> irb
```

You'll see something like this:

```
irb(main):001:0>
```

Most of the pieces of the prompt are unimportant. You'll learn about the parts that are later in this chapter.

Now type a Ruby expression, and press Enter (on Windows) or Return (on Unix-like systems):

```
irb(main):001:0> 1+1
=> 2
irb(main):002:0>
```

irb displays the result and then prompts you to type something more.

2 is the *result* of *evaluating* the expression `1+1`. In the rest of the book you'll be evaluating more exciting expressions, but that's enough for now. Exit from irb like this:

result

```
irb(main):003:0> exit
prompt>
```

2.6 Prompts, Command Lines, Prompts, and irb

There are two kinds of prompts in this book, command-line prompts and irb prompts. If you type a command meant for the command-line interpreter to irb (or vice versa), you'll get confusing results. If what you see on your screen is nothing like what the book tells you to expect, check that you're typing at the right prompt.

If you're typing the command line and should be typing to irb, start irb:

```
prompt> irb
```

If you're typing to irb and should be typing at the command line, exit irb:

```
irb(main):001:0> exit
```