Extracted from:

Everyday Scripting with Ruby

For Teams, Testers, and You

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.

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.

practice files

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 32, 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 69, 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 141, 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 215, 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.

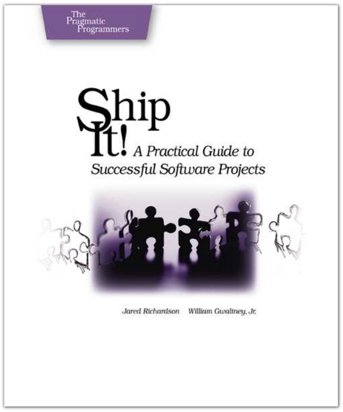
Thank you, those who commented on drafts: Mark Axel, Tracy Beeson, Michael Bolton, Paul Carvalho, Tom Corbett, Bob Corrick, Lisa Crispin, Paul Czyzewski, Shailesh Dongre, Gunjan Doshi, Danny Faught, Zeljko Filipin, Pierre Garique, George Hawthorne, Paddy Healey, Jonathan Kohl, Bhavna Kumar, Walter Kruse, 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.

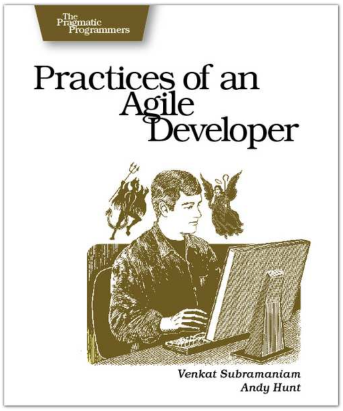
Special thanks to Paul Carvalho for teaching me something I didn't know about Windows and for working through Part IV before Part III, and to Paul Czyzewski for how thoroughly he reviewed the pages I gave him time to review.

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.


The Pragmatic Programmers
Ship It! A Practical Guide to Successful Software Projects
Jared Richardson William Gwaltney, Jr.


The Pragmatic Programmers
Practices of an Agile Developer
Venkat Subramaniam
Andy Hunt