

ZED SHAW'S HARD WAY SERIES



Learn  
the **RUBY**  
**HARD WAY**

THIRD EDITION

A Simple and Idiomatic Introduction  
to the Imaginative World of  
Computational Thinking with Code

ZED A. SHAW

FREE SAMPLE CHAPTER



SHARE WITH OTHERS

# LEARN RUBY THE HARD WAY

Third Edition

# Zed Shaw's Hard Way Series



Visit [informit.com/hardway](http://informit.com/hardway) for a complete list of available publications.

**Z**ed Shaw's **Hard Way Series** emphasizes instruction and *making* things as the best way to get started in many computer science topics. Each book in the series is designed around short, understandable exercises that take you through a course of instruction that creates working software. All exercises are thoroughly tested to verify they work with real students, thus increasing your chance of success. The accompanying video walks you through the code in each exercise. Zed adds a bit of humor and inside jokes to make you laugh while you're learning.



Make sure to connect with us!  
[informit.com/socialconnect](http://informit.com/socialconnect)

**informIT.com**  
the trusted technology learning source

◆ Addison-Wesley

**Safari**  
Video Online

# LEARN RUBY THE HARD WAY

A Simple and Idiomatic Introduction  
to the Imaginative World of  
Computational Thinking with Code

Third Edition

---

**Zed A. Shaw**

◆◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco  
New York • Toronto • Montreal • London • Munich • Paris • Madrid  
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at [corpsales@pearsoned.com](mailto:corpsales@pearsoned.com) or (800) 382-3419.

For government sales inquiries, please contact [governmentsales@pearsoned.com](mailto:governmentsales@pearsoned.com).

For questions about sales outside the United States, please contact [international@pearsoned.com](mailto:international@pearsoned.com).

Visit us on the Web: [informit.com/hardway](http://informit.com/hardway)

*Library of Congress Cataloging-in-Publication Data*

Shaw, Zed, author.

Learn Ruby the hard way : a simple and idiomatic introduction to the imaginative world of computational thinking with code / Zed A. Shaw.—Third edition.

pages cm

Includes index.

ISBN 978-0-321-88499-2 (pbk. : alk. paper)

1. Ruby (Computer program language) I. Title.

QA76.73.R835536 2014

005.1'17—dc23

2014033534

Copyright © 2015 Zed A. Shaw

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-88499-2

ISBN-10: 0-321-88499-X

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.  
First printing, December 2014

# Contents

Preface . . . . .	1
Acknowledgments . . . . .	1
The Hard Way Is Easier . . . . .	2
Reading and Writing . . . . .	2
Attention to Detail . . . . .	2
Spotting Differences . . . . .	3
Do Not Copy-Paste . . . . .	3
Using the Included Videos . . . . .	3
A Note on Practice and Persistence . . . . .	3
A Warning for the Smarties . . . . .	4
<b>Exercise 0 The Setup . . . . .</b>	<b>6</b>
Mac OS X . . . . .	6
OS X: What You Should See . . . . .	7
Windows . . . . .	7
Windows: What You Should See . . . . .	8
Linux . . . . .	8
Linux: What You Should See . . . . .	9
Finding Things on the Internet . . . . .	10
Warnings for Beginners . . . . .	10
<b>Exercise 1 A Good First Program . . . . .</b>	<b>12</b>
What You Should See . . . . .	14
Study Drills . . . . .	16
Common Student Questions . . . . .	17
<b>Exercise 2 Comments and Pound Characters . . . . .</b>	<b>18</b>
What You Should See . . . . .	18
Study Drills . . . . .	18
Common Student Questions . . . . .	19
<b>Exercise 3 Numbers and Math . . . . .</b>	<b>20</b>
What You Should See . . . . .	21
Study Drills . . . . .	21
Common Student Questions . . . . .	22

<b>Exercise 4</b> Variables and Names . . . . .	24
What You Should See . . . . .	25
Study Drills . . . . .	25
Common Student Questions . . . . .	25
<b>Exercise 5</b> More Variables and Printing . . . . .	28
What You Should See . . . . .	28
Study Drills . . . . .	29
Common Student Questions . . . . .	29
<b>Exercise 6</b> Strings and Text . . . . .	30
What You Should See . . . . .	31
Study Drills . . . . .	31
Common Student Question . . . . .	31
<b>Exercise 7</b> More Printing . . . . .	32
What You Should See . . . . .	32
Study Drills . . . . .	33
Common Student Questions . . . . .	33
<b>Exercise 8</b> Printing, Printing . . . . .	34
What You Should See . . . . .	34
Study Drills . . . . .	35
Common Student Questions . . . . .	35
<b>Exercise 9</b> Printing, Printing, Printing . . . . .	36
What You Should See . . . . .	36
Study Drills . . . . .	37
Common Student Questions . . . . .	37
<b>Exercise 10</b> What Was That? . . . . .	38
What You Should See . . . . .	39
Escape Sequences . . . . .	39
Study Drills . . . . .	40
Common Student Questions . . . . .	40
<b>Exercise 11</b> Asking Questions . . . . .	42
What You Should See . . . . .	42
Study Drills . . . . .	43
Common Student Question . . . . .	43
<b>Exercise 12</b> Prompting People for Numbers . . . . .	44
What You Should See . . . . .	44
Study Drills . . . . .	44

<b>Exercise 13</b> Parameters, Unpacking, Variables . . . . .	46
What You Should See . . . . .	46
Study Drills . . . . .	47
Common Student Questions . . . . .	47
<b>Exercise 14</b> Prompting and Passing . . . . .	50
What You Should See . . . . .	50
Study Drills . . . . .	51
Common Student Questions . . . . .	51
<b>Exercise 15</b> Reading Files . . . . .	52
What You Should See . . . . .	53
Study Drills . . . . .	53
Common Student Questions . . . . .	54
<b>Exercise 16</b> Reading and Writing Files . . . . .	56
What You Should See . . . . .	57
Study Drills . . . . .	57
Common Student Questions . . . . .	58
<b>Exercise 17</b> More Files . . . . .	60
What You Should See . . . . .	60
Study Drills . . . . .	61
Common Student Questions . . . . .	61
<b>Exercise 18</b> Names, Variables, Code, Functions . . . . .	62
What You Should See . . . . .	63
Study Drills . . . . .	64
Common Student Questions . . . . .	65
<b>Exercise 19</b> Functions and Variables . . . . .	66
What You Should See . . . . .	67
Study Drills . . . . .	67
Common Student Questions . . . . .	67
<b>Exercise 20</b> Functions and Files . . . . .	70
What You Should See . . . . .	71
Study Drills . . . . .	71
Common Student Questions . . . . .	71
<b>Exercise 21</b> Functions Can Return Something . . . . .	74
What You Should See . . . . .	75
Study Drills . . . . .	75
Common Student Questions . . . . .	76

<b>Exercise 22</b> What Do You Know So Far? . . . . .	78
What You Are Learning . . . . .	78
<b>Exercise 23</b> Read Some Code . . . . .	80
<b>Exercise 24</b> More Practice . . . . .	82
What You Should See . . . . .	83
Study Drills . . . . .	83
Common Student Questions . . . . .	83
<b>Exercise 25</b> Even More Practice . . . . .	84
What You Should See . . . . .	85
Study Drills . . . . .	87
Common Student Questions . . . . .	87
<b>Exercise 26</b> Congratulations, Take a Test! . . . . .	88
Common Student Questions . . . . .	88
<b>Exercise 27</b> Memorizing Logic . . . . .	90
The Truth Terms . . . . .	90
The Truth Tables . . . . .	91
Common Student Question . . . . .	92
<b>Exercise 28</b> Boolean Practice . . . . .	94
What You Should See . . . . .	96
Study Drills . . . . .	96
Common Student Questions . . . . .	96
<b>Exercise 29</b> What If . . . . .	98
What You Should See . . . . .	99
Study Drills . . . . .	99
Common Student Question . . . . .	99
<b>Exercise 30</b> Else and If . . . . .	100
What You Should See . . . . .	101
Study Drills . . . . .	101
Common Student Question . . . . .	101
<b>Exercise 31</b> Making Decisions . . . . .	102
What You Should See . . . . .	103
Study Drills . . . . .	103
Common Student Questions . . . . .	103
<b>Exercise 32</b> Loops and Arrays . . . . .	106
What You Should See . . . . .	108

Study Drills . . . . .	108
Common Student Questions . . . . .	108
<b>Exercise 33 While Loops . . . . .</b>	<b>110</b>
What You Should See . . . . .	111
Study Drills . . . . .	112
Common Student Questions . . . . .	112
<b>Exercise 34 Accessing Elements of Arrays . . . . .</b>	<b>114</b>
Study Drills . . . . .	115
<b>Exercise 35 Branches and Functions . . . . .</b>	<b>116</b>
What You Should See . . . . .	118
Study Drills . . . . .	118
Common Student Questions . . . . .	118
<b>Exercise 36 Designing and Debugging . . . . .</b>	<b>120</b>
Rules for If-Statements . . . . .	120
Rules for Loops . . . . .	120
Tips for Debugging . . . . .	121
Homework . . . . .	121
<b>Exercise 37 Symbol Review . . . . .</b>	<b>122</b>
Keywords . . . . .	122
Data Types . . . . .	124
String Escape Sequences . . . . .	124
Operators . . . . .	125
Reading Code . . . . .	126
Study Drills . . . . .	126
Common Student Question . . . . .	127
<b>Exercise 38 Doing Things to Arrays . . . . .</b>	<b>128</b>
What You Should See . . . . .	129
What Arrays Can Do . . . . .	129
When to Use Arrays . . . . .	130
Study Drills . . . . .	131
Common Student Questions . . . . .	131
<b>Exercise 39 Hashes, Oh Lovely Hashes . . . . .</b>	<b>132</b>
A Hash Example . . . . .	133
What You Should See . . . . .	135
What Hashes Can Do . . . . .	136

Making Your Own Hash Module . . . . .	136
The Code Description . . . . .	140
Three Levels of Arrays . . . . .	142
What You Should See (Again) . . . . .	142
When to Use Hashes or Arrays . . . . .	143
Study Drills . . . . .	143
Common Student Questions . . . . .	144
<b>Exercise 40</b> Modules, Classes, and Objects . . . . .	146
Modules Are Like Hashes . . . . .	146
Classes Are Like Modules . . . . .	148
Objects Are Like Require . . . . .	148
Getting Things from Things . . . . .	150
A First Class Example . . . . .	150
What You Should See . . . . .	151
Study Drills . . . . .	151
Common Student Question . . . . .	151
<b>Exercise 41</b> Learning to Speak Object Oriented . . . . .	152
Word Drills . . . . .	152
Phrase Drills . . . . .	152
Combined Drills . . . . .	153
A Reading Test . . . . .	153
Practice English to Code . . . . .	156
Reading More Code . . . . .	156
Common Student Questions . . . . .	156
<b>Exercise 42</b> Is-A, Has-A, Objects, and Classes . . . . .	158
How This Looks in Code . . . . .	159
Study Drills . . . . .	161
Common Student Questions . . . . .	161
<b>Exercise 43</b> Basic Object-Oriented Analysis and Design . . . . .	164
The Analysis of a Simple Game Engine . . . . .	165
Write or Draw about the Problem . . . . .	165
Extract Key Concepts and Research Them . . . . .	166
Create a Class Hierarchy and Object Map for the Concepts . . . . .	167
Code the Classes and a Test to Run Them . . . . .	168

Repeat and Refine . . . . .	169
Top Down Versus Bottom Up . . . . .	170
The Code for "Gothons from Planet Percal #25" . . . . .	170
What You Should See . . . . .	176
Study Drills . . . . .	177
Common Student Question . . . . .	177
<b>Exercise 44</b> Inheritance Versus Composition . . . . .	178
What Is Inheritance? . . . . .	178
Implicit Inheritance . . . . .	179
Override Explicitly . . . . .	180
Alter Before or After . . . . .	180
All Three Combined . . . . .	182
Using <code>super()</code> with <code>initialize</code> . . . . .	183
Composition . . . . .	183
When to Use Inheritance or Composition . . . . .	185
Study Drills . . . . .	186
Common Student Questions . . . . .	186
<b>Exercise 45</b> You Make a Game . . . . .	188
Evaluating Your Game . . . . .	188
Function Style . . . . .	189
Class Style . . . . .	189
Code Style . . . . .	190
Good Comments . . . . .	190
Evaluate Your Game . . . . .	191
<b>Exercise 46</b> A Project Skeleton . . . . .	192
Creating the Skeleton Project Directory . . . . .	192
Final Directory Structure . . . . .	193
Testing Your Setup . . . . .	195
Using the Skeleton . . . . .	195
Required Quiz . . . . .	195
Common Student Questions . . . . .	196
<b>Exercise 47</b> Automated Testing . . . . .	198
Writing a Test Case . . . . .	198
Testing Guidelines . . . . .	201

What You Should See . . . . .	201
Study Drills . . . . .	202
Common Student Questions . . . . .	202
<b>Exercise 48</b> Advanced User Input . . . . .	204
Our Game Lexicon . . . . .	204
Breaking Up a Sentence . . . . .	205
Lexicon Tuples . . . . .	205
Scanning Input . . . . .	205
Exceptions and Numbers . . . . .	206
A Test First Challenge . . . . .	206
What You Should Test . . . . .	207
Study Drills . . . . .	209
Common Student Questions . . . . .	210
<b>Exercise 49</b> Making Sentences . . . . .	212
Match and Peek . . . . .	212
The Sentence Grammar . . . . .	213
A Word on Exceptions . . . . .	213
The Parser Code . . . . .	213
Playing with the Parser . . . . .	216
What You Should Test . . . . .	217
Study Drills . . . . .	217
Common Student Question . . . . .	217
<b>Exercise 50</b> Your First Website . . . . .	218
Installing Sinatra . . . . .	218
Make a Simple “Hello World” Project . . . . .	219
What’s Happening Here? . . . . .	220
Stopping and Reloading Sinatra . . . . .	221
Fixing Errors . . . . .	221
Create Basic Templates . . . . .	222
Study Drills . . . . .	223
Common Student Questions . . . . .	223
<b>Exercise 51</b> Getting Input from a Browser . . . . .	224
How the Web Works . . . . .	224
How Forms Work . . . . .	226

Creating HTML Forms . . . . .	227
Creating a Layout Template . . . . .	228
Writing Automated Tests for Forms . . . . .	229
Study Drills . . . . .	230
Common Student Question . . . . .	231
<b>Exercise 52</b> The Start of Your Web Game . . . . .	232
Refactoring the Exercise 43 Game . . . . .	232
Sessions and Tracking Users . . . . .	237
Creating an Engine . . . . .	238
Your Final Exam . . . . .	240
<b>Next Steps</b> . . . . .	242
How to Learn Any Programming Language . . . . .	243
<b>Advice from an Old Programmer</b> . . . . .	246
<b>Appendix</b> Command Line Crash Course . . . . .	249
Introduction: Shut Up and Shell . . . . .	249
How to Use This Appendix . . . . .	249
You Will Be Memorizing Things . . . . .	250
The Setup . . . . .	251
Do This . . . . .	251
You Learned This . . . . .	252
Do More . . . . .	252
Paths, Folders, and Directories (pwd) . . . . .	254
Do This . . . . .	255
You Learned This . . . . .	256
Do More . . . . .	256
If You Get Lost . . . . .	256
Do This . . . . .	257
You Learned This . . . . .	257
Make a Directory (mkdir) . . . . .	257
Do This . . . . .	257
You Learned This . . . . .	259
Do More . . . . .	259
Change Directory (cd) . . . . .	260
Do This . . . . .	260

You Learned This . . . . .	263
Do More . . . . .	264
List Directory (ls) . . . . .	264
Do This . . . . .	264
You Learned This . . . . .	269
Do More . . . . .	269
Remove Directory (rmdir) . . . . .	269
Do This . . . . .	270
You Learned This . . . . .	272
Do More . . . . .	272
Moving Around (pushd, popd) . . . . .	273
Do This . . . . .	273
You Learned This . . . . .	275
Do More . . . . .	275
Making Empty Files (Touch, New-Item) . . . . .	276
Do This . . . . .	276
You Learned This . . . . .	277
Do More . . . . .	277
Copy a File (cp) . . . . .	277
Do This . . . . .	277
You Learned This . . . . .	280
Do More . . . . .	281
Moving a File (mv) . . . . .	281
Do This . . . . .	281
You Learned This . . . . .	283
Do More . . . . .	283
View a File (less, MORE) . . . . .	283
Do This . . . . .	284
You Learned This . . . . .	284
Do More . . . . .	284
Stream a File (cat) . . . . .	285
Do This . . . . .	285
You Learned This . . . . .	286
Do More . . . . .	286

---

Removing a File (rm) . . . . .	286
Do This . . . . .	286
You Learned This . . . . .	288
Do More . . . . .	289
Exiting Your Terminal (exit) . . . . .	289
Do This . . . . .	289
You Learned This . . . . .	289
Do More . . . . .	289
Command Line Next Steps . . . . .	290
UNIX Bash References . . . . .	290
PowerShell References . . . . .	290
Index . . . . .	291

*This page intentionally left blank*

# Preface

This simple book is meant to get you started in programming. The title says it's the hard way to learn to write code, but it's actually not. It's only the "hard" way because it uses a technique called *instruction*. Instruction is where I tell you to do a sequence of controlled exercises designed to build a skill through repetition. This technique works very well with beginners, who know nothing and need to acquire basic skills before they can understand more complex topics. It's used in everything from martial arts to music, to even basic math and reading skills.

This book instructs you in Ruby by slowly building and establishing skills through techniques like practice and memorization, then applying them to increasingly difficult problems. By the end of the book you will have the tools needed to begin learning more complex programming topics. I like to tell people that my book gives you your "programming black belt." What this means is that you know the basics well enough to now start learning programming.

If you work hard, take your time, and build these skills, you will learn to code.

## Acknowledgments

I would like to thank Angela for helping me with the first two versions of this book. Without her, I probably wouldn't have bothered to finish it at all. She did the copyediting of the first draft, and supported me immensely while I wrote it.

I also want to thank Rob Sobers for suggesting I make a Ruby version of my Python book and doing the initial work helping me convert it to use Ruby.

I'd also like to thank Greg Newman for doing the original cover art, Brian Shumate for early website designs, and all of the people who read this book and took the time to send me feedback and corrections.

Thank you.

# The Hard Way Is Easier

**W**ith the help of this book, you will do the incredibly simple things that all programmers do to learn a programming language:

1. Go through each exercise.
2. Type in each sample *exactly*.
3. Make it run.

That's it. This will be *very* difficult at first, but stick with it. If you go through this book, and do each exercise for one or two hours a night, you will have a good foundation for moving onto another book about Ruby to continue your studies. This book won't turn you into a programmer overnight, but it will get you started on the path to learning how to code.

This book's job is to teach you the three most essential skills that a beginning programmer needs to know: reading and writing, attention to detail, and spotting differences.

## Reading and Writing

If you have a problem typing, you will have a problem learning to code, and especially if you have a problem typing the fairly odd characters in source code. Without this simple skill you will be unable to learn even the most basic things about how software works.

Typing the code samples and getting them to run will help you learn the names of the symbols, get familiar with typing them, and get you reading the language.

## Attention to Detail

The one skill that separates bad programmers from good programmers is attention to detail. In fact, it's what separates the good from the bad in any profession. You must pay attention to the tiniest details of your work or you will miss important elements of what you create. In programming, this is how you end up with bugs and difficult-to-use systems.

By going through this book, and copying each example *exactly*, you will be training your brain to focus on the details of what you are doing, as you are doing it.

## Spotting Differences

A very important skill (that most programmers develop over time) is the ability to visually notice differences between things. An experienced programmer can take two pieces of code that are slightly different and immediately start pointing out the differences. Programmers have invented tools to make this even easier, but we won't be using any of these. You first have to train your brain the hard way; then use the tools.

While you do these exercises, typing each one in, you will be making mistakes. It's inevitable; even seasoned programmers would make a few. Your job is to compare what you have written to what's required, and fix all the differences. By doing so, you will train yourself to notice mistakes, bugs, and other problems.

## Do Not Copy-Paste

You must *type* each of these exercises in, manually. If you copy and paste, you might as well not even do them. The point of these exercises is to train your hands, your brain, and your mind in how to read, write, and see code. If you copy-paste, you are cheating yourself out of the effectiveness of the lessons.

## Using the Included Videos

*Learn Ruby the Hard Way* has more than five hours of instructional videos to help you with the book. There is one video for each exercise where I either demonstrate the exercise, or give you tips for completing the exercise. The best way to use the videos is if you are stuck when attempting an exercise or for review after you have completed an exercise. This will slowly wean you off of using videos to learn programming and build your skills at understanding code directly. Stick with it, and slowly you won't need the videos, or any videos, to learn programming. You'll be able to just read for the information you need.

## A Note on Practice and Persistence

While you are studying programming, I'm studying how to play guitar. I practice it every day for at least two hours. I play scales, chords, and arpeggios for an hour; and then I learn music theory, ear training, songs, and anything else I can. Some days I study guitar and music for eight hours because I feel like it and it's fun. To me, repetitive practice is natural and just how to learn something. I know that to get good at anything I have to practice every day, even if I suck that day (which is often) or it's difficult. Keep trying and eventually it'll be easier and fun.

Between the time that I wrote *Learn Python the Hard Way* and *Learn Ruby the Hard Way*, I discovered drawing and painting. I fell in love with making visual art at the age of 39; and I have been spending every day studying it in much the same way that I studied guitar, music, and programming. I collected books of instructional material, did what the books said, painted every day, and focused on enjoying the process of learning. I am by no means an “artist,” or even that good, but I can now say that I can draw and paint. The same method I’m teaching you in this book applied to my adventures in art. If you break the problem down into small exercises and lessons, and do them every day, you can learn to do almost anything. If you focus on slowly improving and enjoying the learning process, then you will benefit no matter how good you are at it.

As you study this book, and continue with programming, remember that anything worth doing is difficult at first. Maybe you are the kind of person who is afraid of failure, so you give up at the first sign of difficulty. Maybe you never learned self-discipline, so you can’t do anything that’s “boring.” Maybe you were told that you are “gifted,” so you never attempt anything that might make you seem stupid or not a prodigy. Maybe you are competitive and unfairly compare yourself to someone like me who’s been programming for more than 20 years.

Whatever your reason for wanting to quit, *keep at it*. Force yourself. If you run into a Study Drill you can’t do, or a lesson you just do not understand, then skip it and come back to it later. Just keep going, because with programming there’s this very odd thing that happens. At first, you will not understand anything. It’ll be weird, just like with learning any human language. You will struggle with words, and not know what symbols are what, and it’ll all be very confusing. Then one day *BANG*—your brain will snap and you will suddenly “get it.” If you keep doing the exercises and keep trying to understand them, you will get it. You might not be a master coder, but you will at least understand how programming works.

If you give up, you won’t ever reach this point. You will hit the first confusing thing (which is everything at first) and then stop. If you keep trying—keep typing it in, trying to understand it and reading about it—you will eventually get it. If you go through this whole book, and you still do not understand how to code, at least you gave it a shot. You can say you tried your best and a little more and it didn’t work out, but at least you tried. You can be proud of that.

## A Warning for the Smarties

Sometimes people who already know a programming language will read this book and feel I’m insulting them. There is nothing in this book that is intended to be interpreted as condescending, insulting, or belittling. I simply know more about programming than my *intended* readers. If you think you are smarter than me, then you will feel talked down to and there’s nothing I can do about that because you are not my *intended* reader.

If you are reading this book and flipping out at every third sentence because you feel I'm insulting your intelligence, then I have three points of advice for you:

1. Stop reading my book. I didn't write it for you. I wrote it for people who don't already know everything.
2. Empty before you fill. You will have a hard time learning from someone with more knowledge if you already know everything.
3. Go learn Lisp. I hear people who know everything really like Lisp.

For everyone else who's here to learn, just read everything as if I'm smiling and I have a mischievous little twinkle in my eye.

# A Good First Program

You should have spent a good amount of time in Exercise 0 learning how to install a text editor, run the text editor, run Terminal, and work with both of them. If you haven't done that, then do not go on. You will not have a good time. This is the only time I'll start an exercise with a warning that you should not skip or get ahead of yourself.

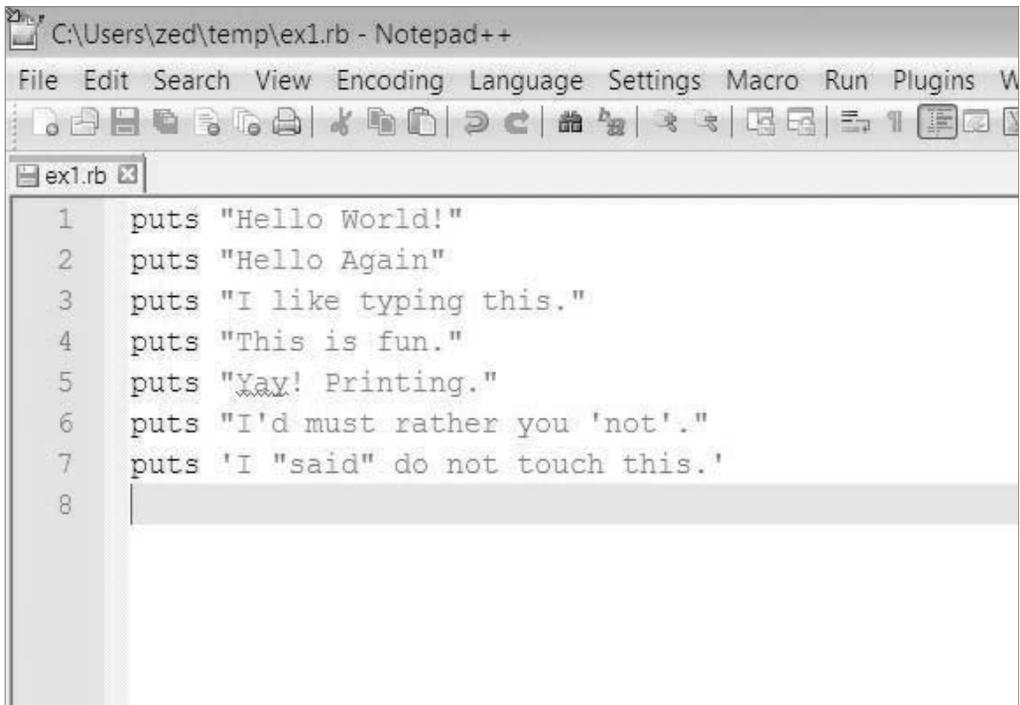
Type the following text into a single file named `ex1.rb`. Ruby works best with files ending in `.rb`.  
`ex1.rb`

```
1 puts "Hello World!"
2 puts "Hello Again"
3 puts "I like typing this."
4 puts "This is fun."
5 puts "Yay! Printing."
6 puts "I'd much rather you 'not'."
7 puts 'I "said" do not touch this.'
```

If you are on Mac OS X, then this is what your text editor might look like if you use TextWrangler:



If you are on Windows using Notepad++, then this is what it would look like:



```
C:\Users\zed\temp\ex1.rb - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins W
ex1.rb
1 puts "Hello World!"
2 puts "Hello Again"
3 puts "I like typing this."
4 puts "This is fun."
5 puts "Yay! Printing."
6 puts "I'd must rather you 'not'."
7 puts 'I "said" do not touch this.'
8
```

Don't worry if your editor doesn't look exactly the same, it should be close though. When you create this file, keep in mind these points:

1. I did not type the line numbers on the left. Those are printed in the book so I can talk about specific lines by saying, "See line 5." You do not type line numbers into Ruby scripts.
2. I have the puts at the beginning of the line and it looks exactly the same as what I have in ex1.rb. Exactly means exactly, not kind of sort of the same. Every single character has to match for it to work. Color doesn't matter, only the characters you type.

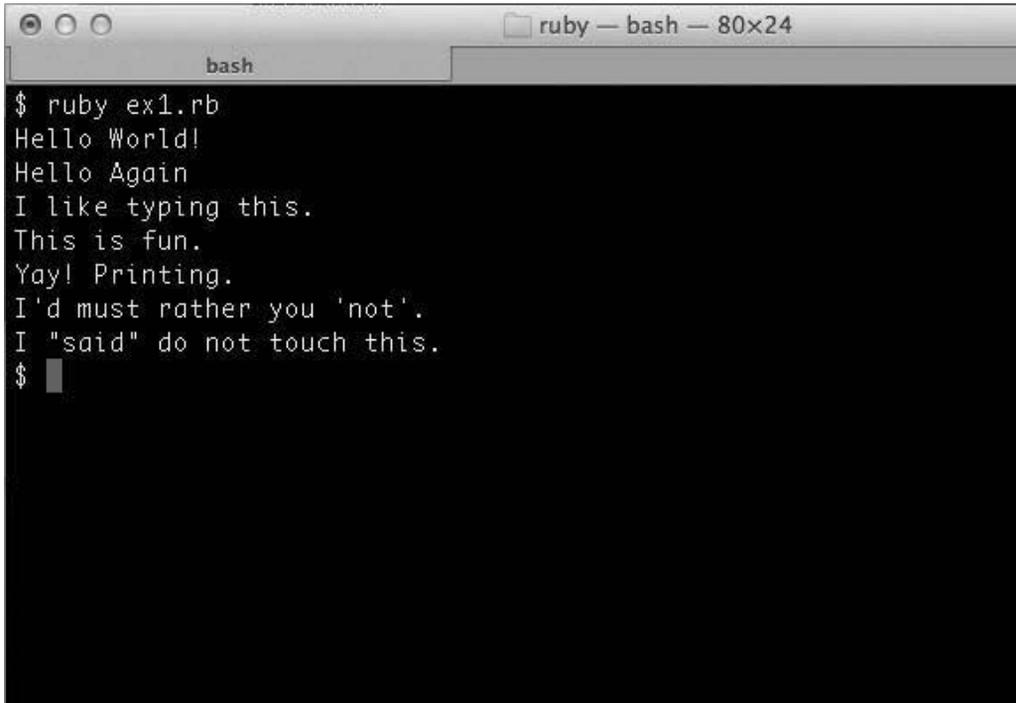
In Terminal *run* the file by typing:

```
ruby ex1.rb
```

If you did it right, then you should see the same output as in the *What You Should See* section of this exercise. If not, you have done something wrong. No, the computer is not wrong.

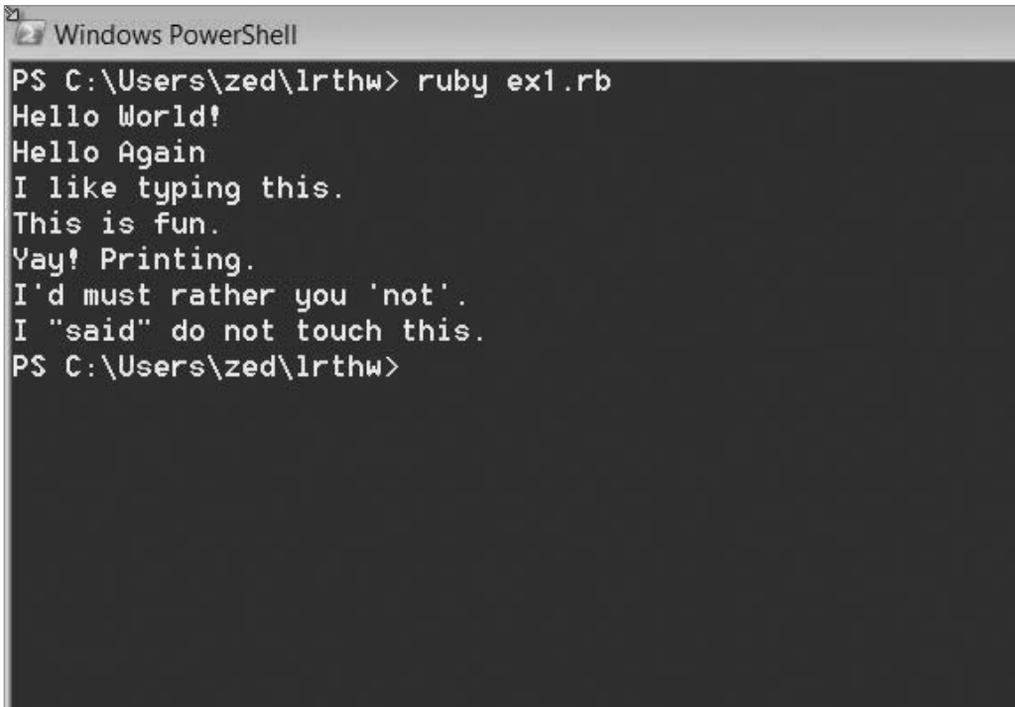
## What You Should See

On Mac OS X in Terminal you should see this:

A screenshot of a Mac OS X Terminal window. The window title bar shows 'ruby — bash — 80x24'. The terminal content shows a prompt '\$' followed by the command 'ruby ex1.rb'. The output consists of several lines of text: 'Hello World!', 'Hello Again', 'I like typing this.', 'This is fun.', 'Yay! Printing.', 'I'd must rather you 'not'.', and 'I "said" do not touch this.'. The prompt '\$' is followed by a cursor block.

```
bash
$ ruby ex1.rb
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd must rather you 'not'.
I "said" do not touch this.
$ █
```

On Windows in PowerShell you should see this:

A screenshot of a Windows PowerShell terminal window. The title bar reads "Windows PowerShell". The prompt is "PS C:\Users\zed\lrthw>". The user has entered the command "ruby ex1.rb". The output of the script is displayed as follows:

```
PS C:\Users\zed\lrthw> ruby ex1.rb
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd must rather you 'not'.
I "said" do not touch this.
PS C:\Users\zed\lrthw>
```

You may see different names before the `ruby ex1.rb` command, but the important part is that you type the command and see the output is the same as mine.

If you have an error, it will look like this:

```
> ruby ex1.rb
ex1.rb:3: syntax error, unexpected tCONSTANT, expecting $end
puts "I like typing this."
```

It's important that you can read these error messages, because you will be making many of these mistakes. Even I make many of these mistakes. Let's look at this line by line.

1. We ran our command in Terminal to run the `ex1.rb` script.
2. Ruby tells us that the file `ex1.rb` has an error on line 3. The type of error is "syntax error," and then some programmer jargon you can usually ignore.
3. It prints the offending line of code for us to see.

---

**WARNING!** If you are from another country, and you get errors about ASCII encodings, then put this at the top of your Ruby scripts:

```
# -*- coding: utf-8 -*-
```

It will fix them so that you can use Unicode UTF-8 in your scripts without a problem.

---

## Study Drills

The Study Drills contain things you should *try* to do. If you can't, skip it and come back later.

For this exercise, try these things:

1. Make your script print another line.
2. Make your script print only one of the lines.
3. Put a `#` (octothorpe) character at the beginning of a line. What did it do? Try to find out what this character does.

From now on, I won't explain how each exercise works unless an exercise is different.

---

**NOTE:** An "octothorpe" is also called a "pound," "hash," "mesh," or any number of names. Pick the one that makes you chill out.

---

## Common Student Questions

These are *actual* questions that real students have asked when doing this exercise.

### **How do you get colors in your editor?**

Save your file first as a `.rb` file, such as `ex1.rb`. Then you'll have color when you type.

### **I get ruby: No such file or directory – ex1.rb (LoadError).**

You need to be in the same directory as the file you created. Make sure you use the `cd` command to go there first. For example, if you saved your file in `1rthw/ex1.rb`, then you would type `cd 1rthw/` before trying to run `ruby ex1.rb`. If you don't know what any of that means, then go through *Appendix A*.

# Comments and Pound Characters

Comments are very important in your programs. They are used to tell you what something does in English, and they are used to disable parts of your program if you need to remove them temporarily. Here's how you use comments in Ruby:

ex2.rb

```
1 # A comment, this is so you can read your program later.
2 # Anything after the # is ignored by ruby.
3
4 puts "I could have code like this." # and the comment after is ignored
5
6 # You can also use a comment to "disable" or comment out a piece of code:
7 # puts "This won't run."
8
9 puts "This will run."
```

From now on, I'm going to write code like this. It is important for you to understand that everything does not have to be literal. Your screen and program may visually look different, but what's important is the text you type into the file you're writing in your text editor. In fact, I could work with any text editor and the results would be the same.

## What You Should See

Exercise 2 Session

```
$ ruby ex2.rb
I could have code like this.
This will run.
```

Again, I'm not going to show you screenshots of all the Terminals possible. You should understand that the above is not a literal translation of what your output should look like visually, but the text between the first \$ ruby ... and last \$ lines will be what you focus on.

## Study Drills

1. Find out if you were right about what the # character does and make sure you know what it's called (octothorpe or pound character).
2. Take your ex2.rb file and review each line going backward. Start at the last line, and check each word in reverse against what you should have typed.

3. Did you find more mistakes? Fix them.
4. Read what you typed out loud, including saying each character by its name. Did you find more mistakes? Fix them.

## Common Student Questions

### **Are you sure # is called the pound character?**

I call it the octothorpe because that is the only name that no country uses, and that works in every country. Every country thinks its name for this one character is both the most important way to do it and the only way it's done. To me, this is simply arrogance and, really, y'all should just chill out and focus on more important things like learning to code.

### **If # is for comments, then how come # -\*- coding: utf-8 -\*- works?**

Ruby still ignores that as code, but it's used as a kind of "hack" or workaround for problems with setting and detecting the format of a file. You will also find a similar kind of comment for editor settings.

### **Why does the # in puts "Hi # there." not get ignored?**

The # in that code is inside a string, so it will be put into the string until the ending " character is hit. These pound characters are just considered characters and aren't considered comments.

### **How do I comment out multiple lines?**

Put a # in front of each one.

### **I can't figure out how to type a # character on my country's keyboard.**

Some countries use the Alt key and combinations of other keys to print characters foreign to their language. You'll have to look online in a search engine to see how to type it.

### **Why do I have to read code backward?**

It's a trick to make your brain not attach meaning to each part of the code, and doing that makes you process each piece exactly. This catches errors and is a handy error-checking technique.

*This page intentionally left blank*

# Command Line Crash Course

This appendix is a quick, super-fast course in using the command line. It is intended to be done rapidly in about a day or two, and is not meant to teach you advanced shell usage.

## Introduction: Shut Up and Shell

This appendix is a crash course in using the command line to make your computer perform tasks. As a crash course, it's not as detailed or extensive as my other books. It is simply designed to get you barely capable enough to start using your computer like a real programmer does. When you're done with this appendix, you will be able to give most of the basic commands that every shell user touches every day. You'll understand the basics of directories and a few other concepts.

The only piece of advice I am going to give you is this:

*Shut up and type all of this in.*

Sorry to be mean, but that's what you have to do. If you have an irrational fear of the command line, the only way to conquer an irrational fear is to just shut up and fight through it.

You are not going to destroy your computer. You are not going to be thrown into some jail at the bottom of Microsoft's Redmond campus. Your friends won't laugh at you for being a nerd. Simply ignore any stupid weird reasons you have for fearing the command line.

Why? Because if you want to learn to code, then you must learn this. Programming languages are advanced ways to control your computer with language. The command line is the little baby brother of programming languages. Learning the command line teaches you to control the computer using language. Once you get past that, you can then move on to writing code and feeling like you actually own the hunk of metal you just bought.

## How to Use This Appendix

The best way to use this appendix is to do the following:

- Get yourself a small paper notebook and a pen.
- Start at the beginning of the appendix and do each exercise exactly as you're told.
- When you read something that doesn't make sense or that you don't understand, *write it down in your notebook*. Leave a little space so you can write an answer.

- After you finish an exercise, go back through your notebook and review the questions you have. Try to answer them by searching online and asking friends who might know the answer. Email me at [help@learncodethehardway.org](mailto:help@learncodethehardway.org) and I'll help you, too.

Just keep going through this process of doing an exercise, writing down questions you have, then going back through and answering the questions you can. By the time you're done, you'll actually know a lot more than you think about using the command line.

## You Will Be Memorizing Things

I'm warning you ahead of time that I'm going to make you memorize things right away. This is the quickest way to get you capable at something, but for some people memorization is painful. Just fight through it and do it anyway. Memorization is an important skill in learning things, so you should get over your fear of it.

Here's how you memorize things:

- Tell yourself you *will* do it. Don't try to find tricks or easy ways out of it, just sit down and do it.
- Write what you want to memorize on some index cards. Put one half of what you need to learn on one side, then the other half on the other side.
- Every day for about 15–30 minutes, drill yourself on the index cards, trying to recall each one. Put any cards you don't get right into a different pile, just drill those cards until you get bored, and then try the whole deck and see if you improve.
- Before you go to bed, drill just the cards you got wrong for about 5 minutes, then go to sleep.

There are other techniques, like you can write what you need to learn on a sheet of paper, laminate it, then stick it to the wall of your shower. While you're bathing, drill the knowledge without looking, and when you get stuck glance at it to refresh your memory.

If you do this every day, you should be able to memorize most of the things I tell you to memorize in about a week to a month. Once you do, nearly everything else becomes easier and intuitive, which is the purpose of memorization. It's not to teach you abstract concepts, but rather to ingrain the basics so that they are intuitive and you don't have to think about them. Once you've memorized these basics, they stop being speed bumps preventing you from learning more advanced abstract concepts.

# The Setup

In this appendix you will be instructed to do three things:

- Do some things in your shell (command line, Terminal, PowerShell).
- Learn about what you just did.
- Do more on your own.

For this first exercise, you'll be expected to get your Terminal open and working so that you can do the rest of the appendix.

## Do This

Get your Terminal, shell, or PowerShell working so you can access it quickly and know that it works.

### Mac OS X

For Mac OS X you'll need to do this:

- Hold down the command key and hit the spacebar.
- In the top right corner, the blue "search bar" will pop up.
- Type: terminal
- Click on the Terminal application that looks kind of like a black box.
- This will open Terminal.
- You can now go to your dock and CTRL-click to pull up the menu, then select Options->Keep In dock.

Now you have your Terminal open and it's in your dock so you can get to it.

### Linux

I'm assuming that if you have Linux, then you already know how to get at your Terminal. Look through the menu for your window manager for anything named "Shell" or "Terminal."

## Windows

On Windows, we're going to use PowerShell. People used to work with a program called `cmd.exe`, but it's not nearly as usable as PowerShell. If you have Windows 7 or later, do this:

- Click Start.
- In "Search programs and files" type: `powershell`
- Hit Enter.

If you don't have Windows 7, you should *seriously* consider upgrading. If you still insist on not upgrading then you can try installing it from Microsoft's download center. Search online to find "powershell downloads" for your version of Windows. You are on your own, though, since I don't have Windows XP, but hopefully the PowerShell experience is the same.

## You Learned This

You learned how to get your Terminal open, so you can do the rest of this appendix.

---

**WARNING!** If you have that really smart friend who already knows Linux, ignore him when he tells you to use something other than bash. I'm teaching you bash. That's it. He will claim that `zsh` will give you 30 more IQ points and win you millions in the stock market. Ignore him. Your goal is to get capable enough and at this level it doesn't matter which shell you use. The next warning is stay off IRC or other places where "hackers" hang out. They think it's funny to hand you commands that can destroy your computer. The command `rm -rf /` is a classic that you *must never type*. Just avoid them. If you need help, make sure you get it from someone you trust and not from random idiots on the Internet.

---

## Do More

This exercise has a large "do more" part. The other exercises are not as involved as this one, but I'm having you prime your brain for the rest of the appendix by doing some memorization. Just trust me: this will make things silky smooth later on.

### Linux/Mac OS X

Take this list of commands and create index cards with the names on the left on one side, and the definitions on the other side. Drill them every day while continuing with the lessons in this appendix.

**pwd** print working directory

**hostname** my computer's network name

**mkdir** make directory

**cd** change directory

**ls** list directory

**rmdir** remove directory

**pushd** push directory

**popd** pop directory

**cp** copy a file or directory

**mv** move a file or directory

**less** page through a file

**cat** print the whole file

**xargs** execute arguments

**find** find files

**grep** find things inside files

**man** read a manual page

**apropos** find which man page is appropriate

**env** look at your environment

**echo** print some arguments

**export** export/set a new environment variable

**exit** exit the shell

**sudo** DANGER! become super user root DANGER!

## Windows

If you're using Windows, then here's your list of commands:

**pwd** print working directory

**hostname** my computer's network name

**mkdir** make directory

**cd** change directory  
**ls** list directory  
**rmdir** remove directory  
**pushd** push directory  
**popd** pop directory  
**cp** copy a file or directory  
**robocopy** robust copy  
**mv** move a file or directory  
**more** page through a file  
**type** print the whole file  
**forfiles** run a command on lots of files  
**dir -r** find files  
**select-string** find things inside files  
**help** read a manual page  
**helpctr** find which manual page is appropriate  
**echo** print some arguments  
**set** export/set a new environment variable  
**exit** exit the shell  
**runas** DANGER! become super user root DANGER!

Drill, drill, drill! Drill until you can say these phrases right away when you see that word. Then drill the inverse, so that you read the phrase and know which command will do that. You're building your vocabulary by doing this, but don't spend so much time you go nuts and get bored.

## Paths, Folders, and Directories (pwd)

In this exercise you learn how to print your working directory with the `pwd` command.

## Do This

I'm going to teach you how to read these "sessions" that I show you. You don't have to type everything I list here, just some of the parts:

- You do *not* type in the \$ (UNIX) or > (Windows). That's just me showing you my session so you can see what I got.
- You type in the stuff after \$ or >, then hit Enter. So if I have \$ pwd, you type just pwd and hit Enter.
- You can then see what I have for output followed by another \$ or > prompt. That content is the output and you should see the same output.

Let's do a simple first command so you can get the hang of this:

### Linux/OS X

Exercise 2 Session

---

```
$ pwd
/Users/zedshaw
$
```

### Windows

Exercise 2 Windows Session

---

```
PS C:\Users\zed> pwd

Path
----
C:\Users\zed

PS C:\Users\zed>
```

---

**WARNING!** In this appendix I need to save space so that you can focus on the important details of the commands. To do this, I'm going to strip out the first part of the prompt (the PS C:\Users\zed above) and leave just the little > part. This means your prompt won't look exactly the same, but don't worry about that.

Remember that from now on I'll include only the > to tell you that's the prompt. I'm doing the same thing for the UNIX prompts, but UNIX prompts are so varied that most people get used to \$ meaning "just the prompt."

---

## You Learned This

Your prompt will look different from mine. You may have your user name before the \$ and the name of your computer. On Windows it will probably look different, too. The key is that you see this pattern:

- There's a prompt.
- You type a command there. In this case, it's `pwd`.
- It printed something.
- Repeat.

You just learned what `pwd` does, which means "print working directory." What's a directory? It's a folder. Folder and directory are the same thing, and they're used interchangeably. When you open your file browser on your computer to graphically find files, you are walking through folders. Those folders are the exact same things as these "directories" we're going to work with.

## Do More

- Type `pwd` 20 times and each time say "print working directory."
- Write down the path that this command gives you. Find it with your graphical file browser of choice.
- No, seriously, type it 20 times and say it out loud. Sssh. Just do it.

## If You Get Lost

As you go through these instructions, you may get lost. You may not know where you are or where a file is and have no idea how to continue. To solve this problem, I am going to teach you the commands to type to stop being lost.

Whenever you get lost, it is most likely because you were typing commands and have no idea where you've ended up. What you should do is type `pwd` to *print your current directory*. This tells you where you are.

The next thing you need is a way of getting back to where you are safe, your home. To do this, type `cd ~` and you are back in your home.

This means if you get lost at any time, you should type:

```
pwd
cd ~
```

The first command `pwd` tells you where you are. The second command `cd ~` takes you home so you can try again.

## Do This

Right now figure out where you are, and then go home using `pwd` and `cd ~`. This will ensure that you are always in the right place.

## You Learned This

How to get back to your home if you ever get lost.

# Make a Directory (`mkdir`)

In this exercise you learn how to make a new directory (folder) using the `mkdir` command.

## Do This

Remember! *You need to go home first!* Do your `pwd` and then `cd ~` before doing this exercise. Before you do *all* exercises in this appendix, always go home first!

Linux/OS X

Exercise 4 Session

---

```
$ pwd
$ cd ~
$ mkdir temp
$ mkdir temp/stuff
$ mkdir temp/stuff/things
$ mkdir -p temp/stuff/things/frank/joe/alex/john
$
```

## Windows

Exercise 4 Windows Session

---

```
> pwd
> cd ~
> mkdir temp
```

```
Directory: C:\Users\zed
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:02 AM		temp

```
> mkdir temp/stuff
```

```
Directory: C:\Users\zed\temp
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:02 AM		stuff

```
> mkdir temp/stuff/things
```

```
Directory: C:\Users\zed\temp\stuff
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		things

```
> mkdir temp/stuff/things/frank/joe/alex/john
```

```
Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex
```

```
Mode                LastWriteTime         Length Name
----                -
d-----           12/17/2011   9:03 AM         john
```

```
>
```

This is the only time I'll list the `pwd` and `cd ~` commands. They are expected in the exercises *every time*. Do them all the time.

## You Learned This

Now we get into typing more than one command. These are all the different ways you can run `mkdir`. What does `mkdir` do? It make directories. Why are you asking that? You should be doing your index cards and getting your commands memorized. If you don't know that "`mkdir` makes directories," then keep working the index cards.

What does it mean to make a directory? You might call directories "folders." They're the same thing. All you did in this exercise is create directories inside directories inside of more directories. This is called a "path" and it's a way of saying "first temp, then stuff, then things, and that's where I want it." It's a set of directions to the computer of where you want to put something in the tree of folders (directories) that make up your computer's hard disk.

---

**WARNING!** In this appendix I'm using the / (slash) character for all paths since it works the same on all computers now. However, Windows users need to know that you can also use the \ (backslash) character and other Windows users will expect that at times.

---

## Do More

- The concept of a "path" might confuse you at this point. Don't worry. We'll do a lot more with them and then you'll get it.
- Make 20 other directories inside the temp directory in various levels. Go look at them with a graphical file browser.
- Make a directory with a space in the name by putting quotes around it: `mkdir "I Have Fun"`
- If the temp directory already exists, then you'll get an error. Use `cd` to change to a work directory that you can control and try it there. On Windows, Desktop is a good place.

## Change Directory (cd)

In this exercise you learn how to change from one directory to another using the `cd` command.

### Do This

I'm going to give you the instructions for these sessions one more time:

- You do *not* type in the `$` (UNIX) or `>` (Windows).
- You type in the stuff after this, then hit Enter. If I have `$ cd temp` you just type `cd temp` and hit Enter.
- The output comes after you hit Enter, followed by another `$` or `>` prompt.
- Always go home first! Do `pwd` and then `cd ~` so you go back to your starting point.

### Linux/OS X

### Exercise 5 Session

---

```
$ cd temp
$ pwd
~/temp
$ cd stuff
$ pwd
~/temp/stuff
$ cd things
$ pwd
~/temp/stuff/things
$ cd frank/
$ pwd
~/temp/stuff/things/frank
$ cd joe/
$ pwd
~/temp/stuff/things/frank/joe
$ cd alex/
$ pwd
~/temp/stuff/things/frank/joe/alex
$ cd john/
$ pwd
~/temp/stuff/things/frank/joe/alex/john
```

```
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things/frank/joe
$ cd ..
$ cd ..
$ pwd
~/temp/stuff/things
$ cd ../../..
$ pwd
~/
$ cd temp/stuff/things/frank/joe/alex/john
$ pwd
~/temp/stuff/things/frank/joe/alex/john
$ cd ../../../../../../../
$ pwd
~/
$
```

## Windows

### Exercise 5 Windows Session

---

```
> cd temp
> pwd
```

Path

----

```
C:\Users\zed\temp
```

```
> cd stuff
> pwd
```

Path

----

```
C:\Users\zed\temp\stuff
```

```
> cd things
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things
```

```
> cd frank
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank
```

```
> cd joe
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank\joe
```

```
> cd alex
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank\joe\alex
```

```
> cd john
```

```
> pwd
```

```
Path
```

```
----
```

```
C:\Users\zed\temp\stuff\things\frank\joe\alex\john
```

```
> cd ..
```

```
> cd ..
> cd ..
> pwd
```

Path

----

```
C:\Users\zed\temp\stuff\things\frank
```

```
> cd ../..
> pwd
```

Path

----

```
C:\Users\zed\temp\stuff
```

```
> cd ..
> cd ..
> cd temp/stuff/things/frank/joe/alex/john
> cd ../../../../../../../
> pwd
```

Path

----

```
C:\Users\zed
```

```
>
```

## You Learned This

You made all these directories in the last exercise, and now you're just moving around inside them with the `cd` command. In my session, I also use `pwd` to check where I am, so remember not to type the output that `pwd` prints. For example, on line 3 you see `~/temp` but that's the output of `pwd` from the prompt above it. *Do not type this in.*

You should also see how I use the `..` to move "up" in the tree and path.

## Do More

A very important part of learning to use the command line interface (CLI) on a computer with a graphical user interface (GUI) is figuring out how they work together. When I started using computers, there was no "GUI" and you did everything with the DOS prompt (the CLI). Later, when computers became powerful enough that everyone could have graphics, it was simple for me to match CLI directories with GUI windows and folders.

Most people today, however, have no comprehension of the CLI, paths, and directories. In fact, it's very difficult to teach it to them and the only way to learn about the connection is for you to constantly work with the CLI until one day it clicks that things you do in the GUI will show up in the CLI.

The way you do this is by spending some time finding directories with your GUI file browser, then going to them with your CLI. This is what you'll do next.

- `cd` to the `joe` directory with one command.
- `cd` back to `temp` with one command, but not further above that.
- Find out how to `cd` to your "home directory" with one command.
- `cd` to your Documents directory, then find it with your GUI file browser (e.g., Finder, Windows Explorer, etc.).
- `cd` to your Downloads directory, then find it with your file browser.
- Find another directory with your file browser, then `cd` to it.
- Remember when you put quotes around a directory with spaces in it? You can do that with any command. For example, if you have a directory `I Have Fun`, then you can do:  
`cd "I Have Fun"`

## List Directory (`ls`)

In this exercise you learn how to list the contents of a directory with the `ls` command.

## Do This

Before you start, make sure you `cd` back to the directory above `temp`. If you have no idea where you are, use `pwd` to figure it out and then move there.

**Linux/OS X****Exercise 6 Session**

---

```
$ cd temp
$ ls
stuff
$ cd stuff
$ ls
things
$ cd things
$ ls
frank
$ cd frank
$ ls
joe
$ cd joe
$ ls
alex
$ cd alex
$ ls
$ cd john
$ ls
$ cd ..
$ ls
john
$ cd ../../../../
$ ls
frank
$ cd ../../
$ ls
stuff
$
```

**Windows****Exercise 6 Windows Session**

---

```
> cd temp
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		stuff

```
> cd stuff
> ls
```

Directory: C:\Users\zed\temp\stuff

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		things

```
> cd things
> ls
```

Directory: C:\Users\zed\temp\stuff\things

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		frank

```
> cd frank
> ls
```

Directory: C:\Users\zed\temp\stuff\things\frank

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		joe

```
> cd joe
> ls
```

Directory: C:\Users\zed\temp\stuff\things\frank\joe

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		alex

```
> cd alex
> ls
```

Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		john

```
> cd john
> ls
> cd ..
> ls
```

Directory: C:\Users\zed\temp\stuff\things\frank\joe\alex

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		john

```
> cd ..  
> ls
```

Directory: C:\Users\zed\temp\stuff\things\frank\joe

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		alex

```
> cd ../../..  
> ls
```

Directory: C:\Users\zed\temp\stuff

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		things

```
> cd ..  
> ls
```

```
Directory: C:\Users\zed\temp

Mode                LastWriteTime         Length Name
----                -
d-----           12/17/2011   9:03 AM         stuff

>
```

## You Learned This

The `ls` command lists out the contents of the directory you are currently in. You can see me use `cd` to change into different directories and then list what's in them so I know which directory to go to next.

There are a lot of options for the `ls` command, but you'll learn how to get help on those later when we cover the `help` command.

## Do More

- *Type every one of these commands in!* You have to actually type these to learn them. Just reading them is *not* good enough. I'll stop yelling now.
- On UNIX, try the `ls -lR` command while you're in `temp`.
- On Windows, do the same thing with `dir -R`.
- Use `cd` to get to other directories on your computer and then use `ls` to see what's in them.
- Update your notebook with new questions. I know you probably have some, because I'm not covering everything about this command.
- Remember that if you get lost, then use `ls` and `pwd` to figure out where you are, then go to where you need to be with `cd`.

## Remove Directory (rmdir)

In this exercise you learn how to remove an empty directory.

## Do This

Linux/OS X

Exercise 7 Session

---

```
$ cd temp
$ ls
stuff
$ cd stuff/things/frank/joe/alex/john/
$ cd ..
$ rmdir john
$ cd ..
$ rmdir alex
$ cd ..
$ ls
joe
$ rmdir joe
$ cd ..
$ ls
frank
$ rmdir frank
$ cd ..
$ ls
things
$ rmdir things
$ cd ..
$ ls
stuff
$ rmdir stuff
$ pwd
~/temp
$
```

---

**WARNING!** If you try to do `rmdir` on Mac OS X and it refuses to remove the directory even though you are *positive* it's empty, then there is actually a file in there called `.DS_Store`. In that case, type `rm -rf <dir>` instead (replace `<dir>` with the directory name).

---

## Windows

## Exercise 7 Windows Session

```
> cd temp
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:03 AM		stuff

```
> cd stuff/things/frank/joe/alex/john/
> cd ..
> rmdir john
> cd ..
> rmdir alex
> cd ..
> rmdir joe
> cd ..
> rmdir frank
> cd ..
> ls
```

Directory: C:\Users\zed\temp\stuff

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/17/2011 9:14 AM		things

```
> rmdir things
> cd ..
> ls
```

```

Directory: C:\Users\zed\temp

Mode                LastWriteTime         Length Name
----                -
d-----           12/17/2011   9:14 AM         stuff

> rmdir stuff
> pwd

Path
----
C:\Users\zed\temp

> cd ..
>

```

## You Learned This

I'm now mixing up the commands so make sure you type them exactly and pay attention. Every time you make a mistake, it's because you aren't paying attention. If you find yourself making many mistakes, then take a break or just quit for the day. You've always got tomorrow to try again.

In this example you'll learn how to remove a directory. It's easy. You just go to the directory right above it, then type `rmdir <dir>`, replacing `<dir>` with the name of the directory to remove.

## Do More

- Make 20 more directories and remove them all.
- Make a single path of directories that is 10 deep and remove them one at a time just like I did.
- If you try to remove a directory with content you will get an error. I'll show you how to remove these in later exercises.

## Moving Around (pushd, popd)

In this exercise you learn how to save your current location and go to a new location with `pushd`. You then learn how to return to the saved location with `popd`.

### Do This

Linux/OS X

Exercise 8 Session

---

```
$ cd temp
$ mkdir -p i/like/icecream
$ pushd i/like/icecream
~/temp/i/like/icecream ~/temp
$ popd
~/temp
$ pwd
~/temp
$ pushd i/like
~/temp/i/like ~/temp
$ pwd
~/temp/i/like
$ pushd icecream
~/temp/i/like/icecream ~/temp/i/like ~/temp
$ pwd
~/temp/i/like/icecream
$ popd
~/temp/i/like ~/temp
$ pwd
~/temp/i/like
$ popd
~/temp
$ pushd i/like/icecream
~/temp/i/like/icecream ~/temp
$ pushd
~/temp ~/temp/i/like/icecream
$ pwd
~/temp
$ pushd
```

```
~/temp/i/like/icecream ~/temp
$ pwd
~/temp/i/like/icecream
$
```

## Windows

Exercise 8 Windows Session

---

```
> cd temp
> mkdir -p i/like/icecream
```

Directory: C:\Users\zed\temp\i\like

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/20/2011 11:05 AM		icecream

```
> pushd i/like/icecream
> popd
> pwd
```

```
Path
----
C:\Users\zed\temp
```

```
> pushd i/like
> pwd
```

```
Path
----
C:\Users\zed\temp\i\like
```

```
> pushd icecream
> pwd
```

```
Path
----
C:\Users\zed\temp\i\like\icecream
```

```
> popd
> pwd
```

```
Path
----
C:\Users\zed\temp\i\like
```

```
> popd
>
```

## You Learned This

You're getting into programmer territory with these commands, but they're so handy I have to teach them to you. These commands let you temporarily go to a different directory and then come back, easily switching between the two.

The `pushd` command takes your current directory and "pushes" it into a list for later, then it *changes* to another directory. It's like saying, "Save where I am, then go here."

The `popd` command takes the last directory you pushed and "pops" it off, taking you back there.

Finally, on UNIX, the command `pushd`, if you run it by itself with no arguments, will switch between your current directory and the last one you pushed. It's an easy way to switch between two directories. *This does not work in PowerShell.*

## Do More

- Use these commands to move around directories all over your computer.
- Remove the `i/like/icecream` directories and make your own, then move around in them.
- Explain to yourself the output that `pushd` and `popd` will print out for you. Notice how it works like a stack?

- You already know this, but remember that `mkdir -p` will make an entire path even if all the directories don't exist. That's what I did very first for this exercise.

## Making Empty Files (Touch, New-Item)

In this exercise you learn how to make an empty file using the `touch` (`new-item` on Windows) command.

### Do This

#### Linux/OS X

---

Exercise 9 Session

```
$ cd temp
$ touch iamcool.txt
$ ls
iamcool.txt
$
```

#### Windows

---

Exercise 9 Windows Session

```
> cd temp
> New-Item iamcool.txt -type file
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	12/17/2011 9:03 AM		iamcool.txt

```
>
```

## You Learned This

You learned how to make an empty file. On UNIX `touch` does this, and it also changes the times on the file. I rarely use it for anything other than making empty files. On Windows, you don't have this command, so you learned how to use the `New-Item` command, which does the same thing but can also make new directories.

## Do More

- UNIX: Make a directory, change to it, and then make a file in it. Then move up one level and run the `rmdir` command in this directory. You *should* get an error. Try to understand why you got this error.
- Windows: Do the same thing, but you won't get an error. You'll get a prompt asking if you really want to remove the directory.

## Copy a File (cp)

In this exercise you learn how to copy a file from one location to another with the `cp` command.

## Do This

Linux/OS X

Exercise 10 Session

---

```
$ cd temp
$ cp iamcool.txt neat.txt
$ ls
iamcool.txt neat.txt
$ cp neat.txt awesome.txt
$ ls
awesome.txt iamcool.txt neat.txt
$ cp awesome.txt thefourthfile.txt
$ ls
awesome.txt iamcool.txt neat.txt thefourthfile.txt
$ mkdir something
$ cp awesome.txt something/
$ ls
awesome.txt iamcool.txt neat.txt something thefourthfile.txt
```

```
$ ls something/
awesome.txt
$ cp -r something newplace
$ ls newplace/
awesome.txt
$
```

## Windows

### Exercise 10 Windows Session

---

```
> cd temp
> cp iamcool.txt neat.txt
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt

```
> cp neat.txt awesome.txt
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	12/22/2011 4:49 PM	0	awesome.txt
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt

```
> cp awesome.txt thefourthfile.txt
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
-----	-----	-----	-----
-a---	12/22/2011 4:49 PM	0	awesome.txt
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt
-a---	12/22/2011 4:49 PM	0	thefourthfile.txt

> mkdir something

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
-----	-----	-----	-----
d----	12/22/2011 4:52 PM		something

> cp awesome.txt something/

> ls

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
-----	-----	-----	-----
d----	12/22/2011 4:52 PM		something
-a---	12/22/2011 4:49 PM	0	awesome.txt
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt
-a---	12/22/2011 4:49 PM	0	thefourthfile.txt

> ls something

```
Directory: C:\Users\zed\temp\something
```

```
Mode                LastWriteTime         Length Name
----                -
-a---             12/22/2011   4:49 PM           0 awesome.txt
```

```
> cp -recurse something newplace
> ls newplace
```

```
Directory: C:\Users\zed\temp\newplace
```

```
Mode                LastWriteTime         Length Name
----                -
-a---             12/22/2011   4:49 PM           0 awesome.txt
```

```
>
```

## You Learned This

Now you can copy files. It's simple to just take a file and copy it to a new one. In this exercise I also make a new directory and copy a file into that directory.

I'm going to tell you a secret about programmers and system administrators now: they are lazy. I'm lazy. My friends are lazy. That's why we use computers. We like to make computers do boring things for us. In the exercises so far you have been typing repetitive boring commands so that you can learn them, but usually it's not like this. Usually if you find yourself doing something boring and repetitive, there's probably a programmer who has figured out how to make it easier. You just don't know about it.

The other thing about programmers is they aren't nearly as clever as you think. If you overthink what to type, then you'll probably get it wrong. Instead, try to imagine what the name of a command is and try it. Chances are that it's a name or some abbreviation similar to what you thought it was. If you still can't figure it out intuitively, then ask around and search online. Hopefully it's not something really stupid like ROBOCOPY.

## Do More

- Use the `cp -r` command to copy more directories with files in them.
- Copy a file to your home directory or desktop.
- Find these files in your graphical user interface and open them in a text editor.
- Notice how sometimes I put a / (slash) at the end of a directory? That makes sure the file is really a directory, so if the directory doesn't exist I'll get an error.

## Moving a File (mv)

In this exercise you learn how to move a file from one location to another using the `mv` command.

### Do This

#### Linux/OS X

---

#### Exercise 11 Session

```
$ cd temp
$ mv awesome.txt uncool.txt
$ ls
newplace uncool.txt
$ mv newplace oldplace
$ ls
oldplace uncool.txt
$ mv oldplace newplace
$ ls
newplace uncool.txt
$
```

#### Windows

---

#### Exercise 11 Windows Session

```
> cd temp
> mv awesome.txt uncool.txt
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/22/2011 4:52 PM		newplace
d----	12/22/2011 4:52 PM		something
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt
-a---	12/22/2011 4:49 PM	0	thefourthfile.txt
-a---	12/22/2011 4:49 PM	0	uncool.txt

```
> mv newplace oldplace
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/22/2011 4:52 PM		oldplace
d----	12/22/2011 4:52 PM		something
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt
-a---	12/22/2011 4:49 PM	0	thefourthfile.txt
-a---	12/22/2011 4:49 PM	0	uncool.txt

```
> mv oldplace newplace
> ls newplace
```

Directory: C:\Users\zed\temp\newplace

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	12/22/2011 4:49 PM	0	awesome.txt

> ls

Directory: C:\Users\zed\temp

Mode	LastWriteTime	Length	Name
----	-----	-----	----
d----	12/22/2011 4:52 PM		newplace
d----	12/22/2011 4:52 PM		something
-a---	12/22/2011 4:49 PM	0	iamcool.txt
-a---	12/22/2011 4:49 PM	0	neat.txt
-a---	12/22/2011 4:49 PM	0	thefourthfile.txt
-a---	12/22/2011 4:49 PM	0	uncool.txt

>

## You Learned This

Moving files or, rather, renaming them. It's easy: give the old name and the new name.

## Do More

- Move a file in the newplace directory to another directory, then move it back.

## View a File (less, MORE)

To do this exercise you're going to do some work using the commands you know so far. You'll also need a text editor that can make plain text (.txt) files. Here's what you do:

- Open your text editor and type some stuff into a new file. On OS X this could be TextWrangler. On Windows this might be Notepad++. On Linux this could be `gedit`. Any editor will work.

- Save that file to your desktop and name it `test.txt`.
- In your shell use the commands you know to *copy* this file to your temp directory that you've been working with.

Once you've done that, complete this exercise.

## Do This

### Linux/OS X

Exercise 12 Session

---

```
$ less test.txt
[displays file here]
$
```

That's it. To get out of `less`, just type `q` (as in quit).

### Windows

Exercise 12 Windows Session

---

```
> more test.txt
[displays file here]
>
```

---

**WARNING!** In the output I'm showing `[displays file here]` to "abbreviate" what that program shows. I'll do this when I mean to say, "Showing you the output of this program is too complex, so just insert what you see on your computer here and pretend I did show it to you." Your screen will not actually show this.

---

## You Learned This

This is one way to look at the contents of a file. It's useful because, if the file has many lines, it will "page" so that only one screenful at a time is visible. In the *Do More* section, you'll play with this some more.

## Do More

- Open your text file again and repeatedly copy-paste the text so that it's about 50–100 lines long.
- Copy it to your temp directory again so you can look at it.

- Now do the exercise again, but this time page through it. On UNIX you use the spacebar and w (the letter w) to go down and up. Arrow keys also work. On Windows, just hit the spacebar to page through.
- Look at some of the empty files you created.
- The cp command will overwrite files that already exist, so be careful when copying files around.

## Stream a File (cat)

You're going to do some more setup for this one so you get used to making files in one program and then accessing them from the command line. With the same text editor from the last exercise, create another file named test2.txt but this time save it directly to your temp directory.

### Do This

#### Linux/OS X

Exercise 13 Session

---

```
$ less test2.txt
[displays file here]
$ cat test2.txt
I am a fun guy.
Don't you know why?
Because I make poems,
that make babies cry.
$ cat test.txt
Hi there this is cool.
$
```

#### Windows

Exercise 13 Windows Session

---

```
> more test2.txt
[displays file here]
> cat test2.txt
I am a fun guy.
Don't you know why?
```

```
Because I make poems,  
that make babies cry.  
> cat test.txt  
Hi there this is cool.  
>
```

Remember that when I say `[displays file here]`, I'm abbreviating the output of that command so I don't have to show you exactly everything.

## You Learned This

Do you like my poem? Totally going to win a Nobel. Anyway, you already know the first command, and I'm just having you check that your file is there. Then you `cat` the file to the screen. This command spews the whole file to the screen with no paging or stopping. To demonstrate that, I have you do this to the `test.txt`, which should just spew a bunch of lines from that exercise.

## Do More

- Make a few more text files and work with `cat`.
- UNIX: Try `cat test.txt test2.txt` and see what it does.
- Windows: Try `cat test.txt, test2.txt` and see what it does.

## Removing a File (rm)

In this exercise you learn how to remove (delete) a file using the `rm` command.

## Do This

Linux/OS X

Exercise 14 Session

---

```
$ cd temp  
$ ls  
uncool.txt iamcool.txt neat.txt something thefourthfile.txt  
$ rm uncool.txt  
$ ls
```

```

iamcool.txt neat.txt something thefourthfile.txt
$ rm iamcool.txt neat.txt thefourthfile.txt
$ ls
something
$ cp -r something newplace
$
$ rm something/awesome.txt
$ rmdir something
$ rm -rf newplace
$ ls
$

```

## Windows

### Exercise 14 Windows Session

---

```

> cd temp
> ls

```

Directory: C:\Users\zed\temp

Mode	LastWriteTime		Length	Name
----	-----		-----	----
d----	12/22/2011	4:52 PM		newplace
d----	12/22/2011	4:52 PM		something
-a---	12/22/2011	4:49 PM	0	iamcool.txt
-a---	12/22/2011	4:49 PM	0	neat.txt
-a---	12/22/2011	4:49 PM	0	thefourthfile.txt
-a---	12/22/2011	4:49 PM	0	uncool.txt

```

> rm uncool.txt
> ls

```

Directory: C:\Users\zed\temp

Mode	LastWriteTime		Length	Name
----	-----		-----	----
d----	12/22/2011	4:52 PM		newplace
d----	12/22/2011	4:52 PM		something
-a---	12/22/2011	4:49 PM	0	iamcool.txt
-a---	12/22/2011	4:49 PM	0	neat.txt
-a---	12/22/2011	4:49 PM	0	thefourthfile.txt

```
> rm iamcool.txt
> rm neat.txt
> rm thefourthfile.txt
> ls
```

Directory: C:\Users\zed\temp

Mode	LastWriteTime		Length	Name
----	-----		-----	----
d----	12/22/2011	4:52 PM		newplace
d----	12/22/2011	4:52 PM		something

```
> cp -r something newplace
> rm something/awesome.txt
> rmdir something
> rm -r newplace
> ls
>
```

## You Learned This

Here we clean up the files from the last exercise. Remember when I had you try to `rmdir` on a directory with something in it? Well, that failed because you can't remove a directory with files in it. Instead you have to remove the file, or recursively delete all of its contents. That's what you did at the end of this example.

## Do More

- Clean up everything in temp from all the exercises so far.
- Write in your notebook to be careful when running recursive remove commands on files.

## Exiting Your Terminal (exit)

### Do This

#### Linux/OS X

Exercise 23 Session

---

```
$ exit
```

#### Windows

Exercise 23 Windows Session

---

```
> exit
```

## You Learned This

Your final exercise is how to exit a Terminal. Again, this is very easy, but I'm going to have you do more.

## Do More

For your last set of exercises, I want you to use the help system to look up a set of commands you should research and learn how to use on your own.

Here's the list for UNIX:

- xargs
- sudo
- chmod
- chown

For Windows, look up these things:

- `forfiles`
- `runas`
- `attrib`
- `icacls`

Find out what these are, play with them, and then add them to your index cards.

## Command Line Next Steps

You have completed the crash course. At this point you should be a barely capable shell user. There's a huge list of tricks and key sequences you don't know yet, and I'm going to give you a few final places to go research more.

## UNIX Bash References

The shell you've been using is called `bash`. It's not the greatest shell but it's everywhere and has a lot of features so it's a good start. Here's a short list of links about `bash` you should visit:

**Bash Cheat Sheet** [http://cli.learncodethehardway.org/bash\\_cheat\\_sheet.pdf](http://cli.learncodethehardway.org/bash_cheat_sheet.pdf) created by Raphael and CC licensed.

**Reference Manual** <http://www.gnu.org/software/bash/manual/bashref.html>

## PowerShell References

On Windows, there's really only PowerShell. Here's a list of useful links related to PowerShell:

**Owner's Manual** <http://technet.microsoft.com/en-us/library/ee221100.aspx>

**Cheat Sheet** <http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=7097>

**Master PowerShell** <http://powershell.com/cs/blogs/ebook/default.aspx>

# Index

## Symbols

" (double quotes)  
 escaping, 38  
 strings and, 28, 33  
 variables and, 30

' (single quotes)  
 escaping, 38  
 strings and, 33  
 variables and, 30

" " " (triple quotes), 37–38

/ (forward-slash) characters, 38–40

\ (backslash) characters, 38–40

\_ (underscore) characters, 24

|| (or) expressions, 91–96

+= (increment by) operators, 71–72, 99

= (equal) characters  
 == vs., 25  
 ARGV and, 46–51  
 in asking questions of users, 42  
 escaping quotes, 38  
 format strings and, 34–37  
 naming variables with, 24–26  
 printing variables with, 28–29  
 returning values from functions with, 74–76  
 setting variables to numbers, 44–45  
 setting variables to strings, 30–33

== (double equal) characters, 25, 92–96

! (not) expressions, 91–96

!= (not equal) characters, 91–96

# (pound) characters, 18–19

## ?? comments, 159–161

#{} (format activators), 28, 35

% (modulus) operators, 22, 35

%{} (format activators), 35

&& (and) expressions, 91–96

\*args (asterisk args), 63–65

@ (object scope), 125, 149–151

[ (left bracket), opening arrays with, 106–108

] (right bracket), closing arrays with, 106–108

> (prompts), 50–51

## Numbers

2-dimensional (2D) arrays, 108

8080 port, 219–223

## A

Addresses, defined, 225

Advanced user input  
 exceptions in, 206  
 game lexicon and, 204–206  
 introduction to, 204  
 lexicon types in, 205  
 numbers in, 206  
 questions about, 210  
 sentence breaks in, 204–205  
 Study Drills on, 209  
 testing first, 206–207  
 testing procedures, 207–209  
 writing scanners in, 205

Advice for programmers, 246–247

After inheritance, 180–182

Algorithms, 140

Analysis  
 of game engines. See Game engine analysis  
 of games, 188–189, 191  
 object-oriented. See Object-oriented analysis  
 top down versus bottom up, 170–176

And (&&) expressions, 91–96

Argument variable (ARGV). See ARGV (argument variable)

ArgumentError, 206

Arguments  
 \*args, 63–65  
 ARGV. See ARGV (argument variable)

- Arguments (*Continued*)
  - arrays and, 128
  - class style and, 189
  - in command line, 46–47
  - in functions, 62–68, 75
- ARGV (argument variable)
  - introduction to, 46–48
  - opening files with, 52–54
  - prompting and passing with, 50–51
- Arrays
  - 2-dimensional, 108
  - accessing elements of, 114–115
  - bucket, 140–142
  - closing, 106–108
  - data structures and, 129–130
  - hashes vs., 132–133, 144
  - levels of, 142–144
  - loops and, 106–112
  - opening, 106–108
  - questions about, 108–109, 131
  - strings and, 128–129
  - Study Drills on, 108, 131
  - when to use, 130–131, 143
- Asking questions of users
  - decision making and, 102–104
  - overview of, 42
  - questions about, 43
  - Study Drills on, 43
- assert commands, 144, 217
- Association, 133–136
- Asterisk args (*\*args*), 63–65
- Automated testing
  - of forms, 229–230
  - guidelines for, 201
  - introduction to, 198
  - questions about, 202
  - results of, 201
  - Study Drills on, 202
  - writing, 198–201
- B**
- Backslash (\) characters, 38–40
- Before inheritance, 180–182
- begin-rescue, 206, 210
- bin/ folders, 192–196
- Bitbucket.com, 80
- Boolean logic
  - overview of, 94–96
  - questions about, 96
  - Study Drills on, 96
  - true/false in, 92
- Branches
  - overview of, 116–119
  - questions about, 119
  - Study Drills on, 118
- Browsers
  - automated testing of forms and, 229–230
  - defined, 225
  - HTML forms and, 226–228
  - input from, generally, 224
  - layout templates and, 228–229
  - questions about, 231
  - Study Drills on, 230–231
  - web workings and, 224–226
  - websites and, 220
- Bucket arrays, 140–142
- C**
- cat (stream file) command, 285–286
- cd (change directory), 260–264
- Chef Solo, 242
- Child classes, 179–186
- chomp, 42–48
- Classes
  - Child, 179–186
  - creating, 148–149
  - in game engine analysis, 167–169
  - hierarchies of, 167–168
  - introduction to, 146–148
  - modules and, 148–149
  - nil and, 160–162
  - in object-oriented programming, generally, 146–151
  - objects vs., 158
  - Parent, 179–186
  - in phrases, 152–153

- style of, 189–190
- testing, 168–169
- `close` command, 56
- Closing arrays with right bracket (`]`), 106–108
- Command line tools
  - `ARGV`. See `ARGV` (argument variable)
  - `cat` (stream file), 285–286
  - `cd` (change directory), 260–264
  - `exit` (exit terminal), 289–290
  - in Linux. See Linux
  - `ls` (list directory), 264–269
  - in Mac OS X. See Mac OS X
  - memorizing, 250
  - `mkdir` (make directory), 257–259
  - `mv` (move file), 281
  - `new-item` (create empty file), 276–277
  - `popd` (return to saved location), 273–276
  - `pushd` (save location, go to new location), 273–276
  - `pwd` command, 254–256
  - references on, 290
  - `rm` (remove file), 286–289
  - `rmdir` (remove directory), 269–272
  - setup for, 251
  - shells, 249
  - `touch` (create empty file), 276–277
  - viewing files with, 283–285
  - for when lost, 256–257
  - in Windows. See Windows
- Comments
  - `## ??` 159–161
  - in English, 18, 24, 33
  - in game creation, 190–191
  - overview of, 18
  - questions about, 18–19
  - Study Drills on, 18–19
- Connections, defined, 225
- Copying files
  - `cp` for, 277–281
  - overview of, 60–61
  - questions about, 61
  - Study Drills on, 61

- Copy-pasting, 3
- Correcting bad code, 88
- `cp` (copy file) command, 277–281

## D

- Data structures, 129–130, 136
- Data types, 124
- Debugging games, 120–121
- Decision making
  - exercises in, 102–103
  - questions about, 103–104
  - Study Drills on, 103
- `def` (define function), 63, 152
- `delete`, 141
- Designing games, 120–121
- Details, significance of, 2
- Dictionaries, 132
- Differences, significance of, 3
- Directories. See Skeleton directories
- Double equal (`==`) characters, 25, 92–96
- Double quotes (`"`). See `"` (double quotes)
- Drawing problems, 165–166

## E

- Ease of learning, 2–5
- Elements of arrays, access to, 114–115
- Eloquent Ruby*, 242
- `else` conditions
  - `elsif as`. See `elsif`
  - making decisions with, 102–104
  - overview of, 100–101
  - questions about, 101
  - Study Drills on, 101
- `elsif`
  - branches and, 116–117
  - definition of, 123
  - introduction to, 101
  - making decisions with, 102–104
  - rules for, 120
- English
  - comments in, 18, 24, 33
  - in object-oriented programming, 156

## Equality operators

- != (not equal), 91–96
- = (single equal). See = (equal) characters
- == (double equal), 25, 92–96
- in Boolean logic, 95–96

## Errors

- debugging games for, 120–121
- exceptions as. See Exceptions
- in if- statements, 100, 120
- load, 17
- name, 25
- ParserError, 213–216
- in prompt variables, 51
- rake test and, 201–202
- reading messages about, 16
- spelling, 37
- syntax. See Syntax errors
- tokens, 205
- in unpacking variables, 47
- in website creation, 221–222

## Escape sequences

- introduction to, 38
- overview of, 38–39
- questions about, 40
- string, 124
- Study Drills on, 40

## Evaluation

- of game engines. See Game engine
- analysis of games, 188–189, 191
- object-oriented. See Object-oriented analysis

## Exceptions

- numbers and, 206
- in sentences, 213
- symbols for, 122–123

exit commands, 116–119, 289–290

Explicitly overriding inheritance, 180

Extracting game concepts, 166. See also Game engine analysis

## F

f variable, 70–71

False

- in Boolean logic, 95–96

- format strings and, 34–35
- math operators and, 20–21
- overview of, 90–92

File.exists?(to\_file), 60–61

## Files

- copying, 60–61, 277–281
- creating empty, 276–277
- functions and, 70–72
- moving, 281
- opening, 52–58
- questions about, 71–72
- reading, 52–58
- reading backward, 24–26
- removing, 286–289
- running, 13, 15
- streaming, 285–286
- Study Drills on, 71
- viewing, 283–285
- writing, 56–58

## First programs

- on Mac OS X, 12, 14
- overview of, 12
- questions about, 17
- Study Drills on, 16
- on Windows, 13, 15

Floating point numbers, 21

## for-loop

- exercises in, 106–108
- questions about, 108–109
- rules for, 120
- Study Drills on, 108
- while-loop vs., 112

Format activators, 28, 35

## Format strings

- = characters and, 34–37
- overview of, 34
- printing, 34
- questions about, 35
- Study Drills on, 35

## Forms

- automated tests for, 229–230
- HTML, 227–231

- layout templates for, 228
- overview of, 226–227
- Forward-slash (/), 38–40
- Frequently asked questions
  - on # (pound) characters, 18–19
  - on advanced user input, 210
  - on arrays, 108–109, 131
  - on asking questions of users, 43
  - on automated testing, 202
  - on Boolean logic, 96
  - on branches, 119
  - on browsers, 231
  - on comments, 18–19
  - on copying files, 61
  - on else conditions, 101
  - on escape sequences, 40
  - on first programs, 17
  - on format strings, 35
  - on functions, generally, 65
  - on functions, values from, 76
  - on functions and files, 71–72
  - on functions and variables, 67–68, 87
  - on has-a phrases, 161–162
  - on hashes, 144
  - on if- statements, 99
  - on inheritance, 186–187
  - on is-a phrases, 161–162
  - on for-loop, 108–109
  - on making decisions, 103–104
  - on math, 22
  - on memorizing logic, 92
  - on names, 25–26
  - on numbers, 22
  - on object-oriented analysis, 177
  - on object-oriented programming, 151, 156
  - on practicing code, 83, 87
  - on printing, 33, 37
  - on prompting and passing, 51
  - on reading and writing files, 54, 58
  - on sentences, 217
  - on skeleton directories, 195–196
  - on strings, 31
  - on symbols, 127

- on text, 31
- on variables, 25–26, 47–48
- on website creation, 223
- on while-loop, 112
- Functions
  - branches and, 115–119
  - checklists for, 64, 70
  - code and, 62–65
  - def (define function) for, 152
  - files and, 70–72
  - in game creation, 189
  - importing and running, 84–86
  - names and, 62–65
  - overview of, 62–64
  - questions about, 65, 71–72, 76
  - Study Drills on, 64, 71, 75–76
  - style of, 189
  - values and, 74–76
  - variables and, 62–68, 84–87

## G

- Game engine analysis. *See also* Games
  - class hierarchies in, 167–168
  - coding classes in, 168
  - extracting/researching concepts in, 166
  - object maps in, 167–168
  - refining code in, 169–170
  - testing classes in, 168–169
  - top down versus bottom up approach to, 170–176
  - writing/drawing problems in, 165–166
- Game engines
  - analysis of. *See* Game engine analysis
  - creation of, 238–240
  - refactoring of, 232–237
  - session-based, 238–241
- Game lexicon
  - exceptions in, 206
  - introduction to, 204
  - lexicon types in, 205
  - numbers in, 206
  - sentence breaks in, 204–205
  - writing scanners and, 205

## Games

- analysis of engines for. *See* Game engine analysis
  - arrays in, 129–131
  - branches in, 116–119
  - class style in, 189–190
  - code style in, 190
  - comments in, 190–191
  - debugging, 120–121
  - designing, 120–121
  - engines for. *See* Game engines
  - evaluating, 188–189, 191
  - function style in, 189
  - functions in, generally, 116–119
  - introduction to, 188
  - lexicon in. *See* Game lexicon
  - user input in. *See* Advanced user input on the web. *See* Web games
- gedit text editor
- setup and, 8–9, 11
  - viewing files in, 283
- gem, 218–219
- get commands, 141
- gets.chomp
- asking questions of users with, 42–48
  - opening files with, 52–54
  - prompting and passing with, 50–51
- Github.com, 80
- Gitorious.org, 80
- Global variables, 68
- Google, 10
- “Gothons from Planet Percal #25,” 170–176
- Grammar, 213

## H

- Handlers, 222–223, 227
- Hard coding, 52
- Hard way overview
  - copy-pasting vs., 3
  - details, significance of, 2
  - differences, significance of, 3
  - ease of learning in, 2
  - instructional videos, 3

- persistence in, 3–4
- practice in, 3
- reading/writing Ruby in, 2
- warnings for programmers in, 4–5

## has-a phrases

- introduction to, 158–159
- overview of, 159–161
- questions about, 161–162
- Study Drills on, 161

## Hashes

- arrays and, 132–133, 142–144
- code description for, 140–142
- as data structures, 136
- exercises in, 133–136
- hash\_key, 141
- introduction to, 132–133
- modules and, 136–139, 146–148
- questions about, 144
- Study Drills on, 143–144
- when to use, 143

“Hello World” project, 219–220. *See also* Websites

## HTML (HyperText Markup Language)

- forms in, 225–231
- session-based game engines in, 238–241
- websites created in, 222–223

## I

- Idiomatic Ruby, 11
- if- statements
  - arrays and, 106
  - else conditions and, 100–101
  - exercises in, 98–99
  - making decisions with, 102–104
  - questions about, 99
  - rules for, 120
  - Study Drills on, 99
- Implicit inheritance, 179
- Increment by (+) operators, 71–72, 99
- Index.GET, 221–224
- Inheritance
  - altering before/after, 180–182
  - combining types of, 182–183

- composition vs., 183–186
- definition of, 178–179
- implicit, 179
- introduction to, 178
- overriding explicitly, 180
- questions about, 186–187
- Study Drills on, 186
- `super()` with `initialize`, 183
- when to use, 185–186

`initialize`, 183

Input methods, 42–47

Instantiating classes, 148–149

Instructional videos, 3

`Integer()` functions, 206

Intended readers, 4

International programming, 16

Internet

- browsers and. See Browsers
- games on. See Web games
- searching, 10

Irb interpreter, 53–54, 85–86

`is-a` phrases

- exercises in, 159–161
- introduction to, 158–159
- questions about, 161–162
- Study Drills on, 161

## K

Keywords, 122–123

## L

Launchpad.net, 80

Layout templates, 228–229

*Learn C the Hard Way*, 242

*Learn Python the Hard Way*,  
4, 242

*Learn Ruby the Hard Way*, 3–4

Learning programming languages, 243–244

Learning Ruby, overview. See Hard way  
overview

Left bracket (`[]`), opening arrays with,  
106–108

Lexicon. See Game lexicon

## Linux

`cat` (stream file) in, 285

`cd` (change directory) in, 260–261

command line tools in, 252–253

`cp` (copy file) in, 277–278

`exit` (exit terminal) in, 289

`less` command in, 284

`ls` (list directory) in, 265

`mkdir` (make directory) in, 257

`mv` (move file) in, 281

`popd` (return to saved location) in,  
273–274

`pushd` (save location, go to new location)  
in, 273–274

`pwd` command in, 255

`rm` (remove file) in, 286–287

`rmdir` (remove directory) in, 270

setup on, 8–10

Terminal setup in, 251

`touch` (create empty file) in, 276–277

Lisp, 5

`list`, 141

List directory (`ls`), 264–269

Localhost, 220–225

Loops

- arrays and, 106–112

- `for-loop`, 106–108, 112, 120

- `while-loop`, 110–112, 120, 169

`ls` (list directory), 264–269

## M

Mac OS X

`cat` (stream file) in, 285

`cd` (change directory) in, 260–261

command line tools in, generally, 252–253

`cp` (copy file) in, 277–278

`exit` (exit terminal) in, 289

first programs on, 12, 14

`less` command in, 284

`ls` (list directory) in, 265

`mkdir` (make directory) in, 257

`mv` (move file) in, 281

`popd` (return to saved location) in, 273–274

- Mac OS X (*Continued*)
    - pushd (save location, go to new location) in, 273–274
    - pwd command in, 255
    - rmdir (remove directory) in, 270
    - running files on, 14
    - setup on, 6–7
    - Terminal setup in, 251
    - touch (create empty file) in, 276–277
  - Make directory (mkdir), 257–259
  - Mapping
    - of hashes, 133–136
    - of objects, 167–168
    - in refactoring, 232–237
  - Match and peek, 212–213
  - Math. *See also* Numbers
    - algorithms for data structures, 140
    - overview of, 20–21
    - questions about, 22
    - Study Drills on, 21
  - Mechanize, 242
  - Memorization
    - of characters, 78
    - of command line tools, 250
    - of logic, 90–92
    - of truth tables, 91–92
    - of truth terms, 90–91
  - mkdir (make directory), 257–259
  - Modules
    - classes and, 148
    - hashes and, 136–139, 146–148
    - in object-oriented programming, generally, 146–151
  - Modulus (%) operators, 22, 35
  - mv (move file) command, 281
- N**
- Names
    - functions and, 62–65
    - overview of, 24–25
    - questions about, 25–26
    - Study Drills on, 25
  - new (create initializer) command, 140
  - new-item (create empty file) command, 276–277
  - nil
    - classes and, 160–162
    - as data type, 124
    - hashes and, 132–139
  - Not (!) expressions, 91–96
  - Not equal (!=) characters, 91–96
  - Notepad++
    - first programs in, 13
    - setup and, 7
    - viewing files in, 283
  - Numbers. *See also* Math
    - in game lexicon, 206
    - overview of, 20–21
    - questions about, 22
    - Study Drills on, 21
- O**
- Object maps, 167–168
  - Object scope (@), 125
  - Object-oriented analysis
    - introduction to, 164–165
    - questions about, 177
    - of simple game engines. *See* Game engine analysis
    - Study Drills on, 176–177
    - top down versus bottom up approach in, 170–176
  - Object-oriented programming (OOP)
    - analysis in. *See* Object-oriented analysis
    - classes in, 148–149
    - english option in, 156
    - examples of, 150–151
    - exercises in, 152
    - getting things from things in, 150
    - hashes in, 146–148
    - introduction to, 148–149
    - modules in, 146–149
    - objects in, 148–149
    - phrase drills for, 152–153
    - questions about, 151, 156
    - reading code in, 153–156

require in, 148–149  
 Study Drills on, 151  
 word drills for, 152  
**Objects**  
   @ for, 125  
   classes vs., 158–161  
   instantiating, 148–149  
   maps of, 167–168  
   in object-oriented programming, generally,  
     146–151  
   require and, 148–149  
**Octothorpe**, 18–19  
**OED** (*Oxford English Dictionary*), 136  
**OOP** (object-oriented programming). See  
   Object-oriented programming (OOP)  
**open** (filename), 52–58  
**Opening arrays with left bracket** ([], 106–108  
**Operators**, defined, 125. See also *specific  
   operators*  
**Or** (| |) expressions, 91–96  
**Overriding inheritance**, 180  
*Oxford English Dictionary* (OED), 136  
**P**  
**Padrino**, 242  
**Parameters**, 46–47  
**Parent classes**, 179–186  
**Parentheses Exponents Multiplication Division  
   Addition Subtraction** (PEMDAS), 22  
**Parsers**, 213–216  
**PEMDAS** (Parentheses Exponents  
   Multiplication Division Addition  
   Subtraction), 22  
**Persistence**, 3–4  
**Phrase drills**, 152–153  
**popd** (return to saved location), 273–276  
**Port 8080**, 219–223  
**Pound** (#) characters. See # (pound) characters  
**PowerShell**  
   first programs in, 15–16  
   references on, 290  
   running files in, 15  
   setup and, 6–8

**Practicing code**  
   exercises in, 82–86  
   importance of, 3  
   questions about, 83, 87  
   Study Drills on, 83, 87  
**Printing**  
   format strings, 34–35  
   practice exercises in, 32–33, 36–37  
   print for, 42  
   print\_ two for, 62  
   pwd (print working directory) for, 254–256  
   questions about, 33, 37  
   Study Drills on, 33, 37  
   variables, 28–29  
**Programmers**  
   advice for, 246–247  
   bad versus good, 86  
   details, significance of, 2  
   differences, significance of, 3  
   warnings for, 4–5  
**Programming languages**, 243–244  
**Project skeleton directories**. See **Skeleton  
   directories**  
**Prompting**  
   > for, 50–51  
   for numbers, 44  
   passing and, 50–51  
   questions about, 51  
   Study Drills on, 44, 51  
**pushd** (save location, go to new location),  
   273–276  
**puts** command  
   comments and, 18–19  
   format strings and, 34–36  
   introduction to, 12–13  
   math symbols and, 20–21  
   strings and, 30–32  
**pwd** (print working directory), 254–256

**Q**

**Questions asked by students**. See **Frequently  
   asked questions**

Questions asked of users. See Asking questions of users

## R

rake test

- automated testing with, 199–202
- setting up, 194
- syntax errors in, 202
- websites and, 230, 232

Rakefile, 193–194, 199–202

.rb suffix, 12

Reading code

- for files. See Reading files
- in object-oriented programming, 153–156
- resources for, 80–81
- in Ruby, generally, 2
- symbols in, 126

Reading files

- backward, 24–26
- exercises in, 52–53, 56–57
- questions about, 54, 58
- read command for, 56
- readline command for, 56
- Study Drills on, 53, 57–58

Refactoring game engines, 232–237

References, 242, 290

Refining code, 169–170

Remove directory (`rmdir`), 269–272

Remove file (`rm`), 286–289

Requests, defined, 225–226

require, 148–149

Researching game concepts, 166. See also

Game engine analysis

Responses, defined, 226

Return to saved location (`popd`), 273–276

Right bracket (`]`), closing arrays with, 106–108

`rm` (remove file), 286–289

`rmdir` (remove directory), 269–272

Room class, 232–237

Ruby, introduction to

- first programs in, 12–17
- idiomatic Ruby, 11

learning, generally. See Hard way overview

setting up, 6–11

Ruby on Rails, 242

RubyMotion, 242

Ruby-Processing, 242

Rules

- for `if`-statements, 120
- for `for`-loop, 120
- for `while`-loop, 120

Running files, 13, 15

## S

Save location, go to new location (`pushd`), 273–276

Scanners, 205

Searching Internet, 10

Sentences

- breaks in, 204–205
- creating, generally, 212
- exceptions in, 213
- grammar in, 213
- match and peek in, 212–213
- parsers in, 213–216
- questions about, 217
- Study Drills on, 217
- testing of, 217

Servers, defined, 226

Session-based game engines, 238–241

Sessions, 237–238

set, 141–142

Setup

- for command line tools, 251
- generally, 6
- Internet searches for, 10
- on Linux, 8–10
- on Mac OS X, 6–7
- warnings about, 10–11
- on Windows, 7–8

Shells, 249

Sinatra

- browser interactions with, 220
- errors in, 221–222
- “Hello World” project in, 219–220

- installing, 218–219
- stopping/reloading, 221
- templates in, 222–223
- Single equal (=) characters. *See* = (equal) characters
- Skeleton directories
  - creating, 192–193
  - final structure of, 193–194
  - introduction to, 192
  - questions about, 195–196
  - quiz on, 195
  - testing setup of, 194
  - using, 195
- Sourceforge.net, 80
- Stateless sessions, 237
- Stream file (cat) command, 285–286
- String escape sequences, 124. *See also* Escape sequences
- Strings
  - arrays and, 128–129
  - escape sequences and, 38
  - format, 34–35
  - questions about, 31
  - Study Drills on, 31
  - text and, 30–31
  - variables and, 28–29
- Student questions. *See* Frequently asked questions
- Study Drills
  - on # characters, 18–19
  - on accessing elements of arrays, 115
  - on advanced user input, 209
  - on arrays, 108, 131
  - on asking questions of users, 43
  - on automated testing, 202
  - on Boolean logic, 96
  - on branches, 118
  - on browsers, 230–231
  - on comments, 18–19
  - on copying files, 61
  - on else conditions, 101
  - on escape sequences, 40
  - on first programs, 16
  - on functions, generally, 64
  - on functions and files, 71
  - on functions and values, 75–76
  - on functions and variables, 67, 87
  - on has-a phrases, 161
  - on hashes, 143–144
  - on if- statements, 99
  - on inheritance, 186
  - on is-a phrases, 161
  - on for-loop, 108
  - on making decisions, 103
  - on math, 21
  - on names, 25
  - on numbers, 21
  - on object-oriented analysis, 176–177
  - on object-oriented programming, 151
  - on practicing code, 83, 87
  - on printing, 29, 33, 37
  - on prompting and passing, 51
  - on prompting for numbers, 44
  - on reading files, 53, 57–58
  - on sentences, 217
  - on strings, 31
  - on symbols, 126–127
  - on text, 31
  - on variables, 25, 29, 47
  - on website creation, 223
  - on while-loop, 112
  - on writing files, 57–58
- Style, 189–190
- sudo gem install, 218–221, 229
- super() with initialize, 183
- super(name), 160–162
- Symbols. *See also specific symbols*
  - data types, 124
  - introduction to, 122
  - keywords, 122–123
  - operators, 125
  - questions about, 127
  - reading code and, 126
  - string escape sequences, 124
  - Study Drills on, 126–127

- Syntax errors
  - branches and, 118
  - defined, 16
  - in `if`-statements, 100
  - `rake test` and, 202
- T**
- Templates, 222–223, 228–229
- Terminal
  - Boolean logic in, 94
  - first programs in, 13–16
  - running files in, 14
  - setup and, 6–11
- Testing
  - automated. *See* Automated testing
  - classes, in game engine analysis, 168–169
  - correcting bad code and, 88
  - procedures for, 207–209
  - `rake` for. *See* `rake test`
  - in sentence creation, 217
  - of skeleton directory setup, 194
  - user input and, 206–209
- Text
  - questions about, 31
  - strings and, 30–31
  - Study Drills on, 31
- Text editors
  - creating, 56–58
  - `gedit`, 8–11, 283
  - Notepad++ as. *See* Notepad++
  - running files on, 12–17
  - setting up, 6–11
  - Terminal as. *See* Terminal
  - TextWrangler as. *See* TextWrangler
- TextWrangler
  - first programs in, 12
  - setup and, 6, 11
  - viewing files in, 283
- Thor, 242
- Top down versus bottom up analysis, 170–176.
  - See also* Object-oriented analysis
- `touch` (create empty file) command, 276–277
- Tracking sessions/users, 237–238
- Triple quotes (" " "), 37–38
- True
  - format strings and, 34–35
  - math operators and, 20–21
  - strings and, 31
- `truncate` command, 56–58
- Truth tables, 91–92
- Truth terms, 90–91
- `txt = open(filename)`, 52–54
- Typing, importance of, 2
- U**
- Underscore (`_`) characters, 24
- UNIX Bash, 290
- Unpacking arguments/variables, 44, 46–48
- Upgrades, 240–241
- URLs, 220–225
- Users
  - asking questions of, 42–43, 102–104
  - input of. *See* Advanced user input
  - tracking, 237–238
- V**
- Values, 74–76, 121
- Variables
  - ARGV. *See* ARGV (argument variable)
  - `f`, 70–71
  - functions and, 62–68, 84–87
  - global, 68
  - naming, 24–26
  - overview of, 24–25
  - passing to scripts, 46–47
  - printing, 28–29
  - questions about, 25–26, 47–48, 67–68
  - setting to numbers, 44–45
  - setting to strings, 30–33
  - strings and, 28–33
  - Study Drills on, 25, 47, 67
- Videos, 3
- Viewing files, 283–285

**W**`'w, '` 56–61

Warnings, 4–5, 10–11

Web games

- engine creation for, 238–240
- introduction to, 232
- refactoring engines for, 232–237
- sessions in, 237–238
- tracking users in, 237–238
- upgrades to, 240–241

Websites

- browsers and. *See* Browsers
  - creating, generally, 218
  - error repair in, 221–222
  - “Hello World” project, 219–220
  - questions about, 223
  - Sinatra for, installing, 218–219
  - Sinatra for, stopping/reloading, 221
  - Study Drills on, 223
  - template creation in, 222–223
- The Well-Grounded Rubyist*, 242
- `while true`, 116–119
- `while-loop`
- exercises in, 110–112
  - in game creation, 169
  - `for-loop` vs., 112
  - questions about, 112
  - rules for, 120
  - Study Drills on, 112

Windows

- `cat` (stream file) in, 285–286
- `cd` (change directory) in, 261–263

- command line tools in, 253–254
  - `cp` (copy file) in, 278–280
  - `exit` (exit terminal) in, 285–286
  - first programs on, 13, 15
  - `ls` (list directory) in, 265–269
  - `mkdir` (make directory) in, 258–259
  - more command in, 284
  - `mv` (move file) in, 281–283
  - `new-item` (create empty file) in, 276–277
  - `popd` (return to saved location), 274–275
  - PowerShell setup in, 252
  - `pushd` (save location, go to new location), 274–275
  - `pwd` command in, 255
  - `rm` (remove file) in, 287–288
  - `rmdir` (remove directory) in, 271–272
  - running files on, 15
  - setup on, 7–8
- Word drills, 152
- Writing files
- exercises in, 56–57
  - questions about, 58
  - Study Drills on, 57–58
  - `write('stuff')` command for, 56
- Writing problems, 165–166
- Writing scanners, 205
- Writing skills, importance of, 2

**Z**

Zen koans, 158

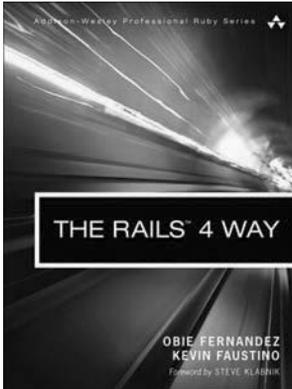
*This page intentionally left blank*



# The Addison-Wesley Professional Ruby Series

[informit.com/ruby](http://informit.com/ruby)

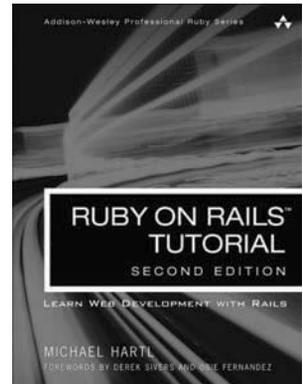
**Expert reference books, eBooks, and videos  
from experienced practitioners**



ISBN-13: 978-0-321-94427-6



ISBN-13: 978-0-321-72133-4



ISBN-13: 978-0-321-83205-4



ISBN-13: 978-0-321-58410-6



ISBN-13: 978-0-13-303999-3



ISBN-13: 978-0-132-80826-2



Look for these titles and more on [informit.com/ruby](http://informit.com/ruby).

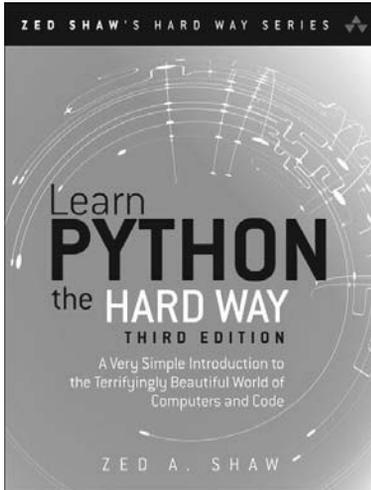
Available in print and eBook formats

ALWAYS LEARNING

PEARSON

# You *Will* Learn Python!

[informit.com/hardway](http://informit.com/hardway)



Zed Shaw has perfected the world's best system for learning Python. Follow it and you will succeed—just like the hundreds of thousands of beginners Zed has taught to date! You bring the discipline, commitment, and persistence; the author supplies everything else.

In *Learn Python the Hard Way, Third Edition*, you'll learn Python by working through 52 brilliantly

crafted exercises. Read them. Type their code *precisely*. (No copying and pasting!) Fix your mistakes. Watch the programs run. As you do, you'll learn how software works; what good programs look like; how to read, write, and think about code; and how to find and fix your mistakes using tricks professional programmers use.

This tutorial will reward you for every minute you put into it. Soon, you'll know one of the world's most powerful, popular programming languages. You'll be a Python programmer.

Watch Zed, too! The accompanying DVD contains 5+ hours of passionate, powerful teaching: a complete Python video course!

---

**informIT**

For more information and sample content visit [informit.com/hardway](http://informit.com/hardway).

**Safari**  
Books Online

eBook and print formats available

**PEARSON**

InformIT is a brand of Pearson and the online presence for the world's leading technology publishers. It's your source for reliable and qualified content and knowledge, providing access to the leading brands, authors, and contributors from the tech community.

▼ Addison-Wesley Cisco Press IBM Press. Microsoft Press

PEARSON  
IT CERTIFICATION

PRENTICE  
HALL

QUE

SAMS

vmware PRESS

## LearnIT at InformIT

Looking for a book, eBook, or training video on a new technology? Seeking timely and relevant information and tutorials. Looking for expert opinions, advice, and tips? **InformIT has a solution.**

- Learn about new releases and special promotions by subscribing to a wide variety of monthly newsletters. Visit [informit.com/newsletters](http://informit.com/newsletters).
- FREE Podcasts from experts at [informit.com/podcasts](http://informit.com/podcasts).
- Read the latest author articles and sample chapters at [informit.com/articles](http://informit.com/articles).
- Access thousands of books and videos in the Safari Books Online digital library. [safari.informit.com](http://safari.informit.com).
- Get Advice and tips from expert blogs at [informit.com/blogs](http://informit.com/blogs).

Visit [informit.com](http://informit.com) to find out all the ways you can access the hottest technology content.

### Are you part of the **IT** crowd?

Connect with Pearson authors and editors via RSS feeds, Facebook, Twitter, YouTube and more! Visit [informit.com/socialconnect](http://informit.com/socialconnect).



# REGISTER



## THIS PRODUCT

[informit.com/register](http://informit.com/register)

Register the Addison-Wesley, Exam Cram, Prentice Hall, Que, and Sams products you own to unlock great benefits.

To begin the registration process, simply go to **[informit.com/register](http://informit.com/register)** to sign in or create an account.

You will then be prompted to enter the 10- or 13-digit ISBN that appears on the back cover of your product.

Registering your products can unlock the following benefits:

- Access to supplemental content, including bonus chapters, source code, or project files.
- A coupon to be used on your next purchase.

Registration benefits vary by product. Benefits will be listed on your Account page under Registered Products.

### **About InformIT — THE TRUSTED TECHNOLOGY LEARNING SOURCE**

INFORMIT IS HOME TO THE LEADING TECHNOLOGY PUBLISHING IMPRINTS Addison-Wesley Professional, Cisco Press, Exam Cram, IBM Press, Prentice Hall Professional, Que, and Sams. Here you will gain access to quality and trusted content and resources from the authors, creators, innovators, and leaders of technology. Whether you're looking for a book on a new technology, a helpful article, timely newsletters, or access to the Safari Books Online digital library, InformIT has a solution for you.

**informIT.com**

THE TRUSTED TECHNOLOGY LEARNING SOURCE

Addison-Wesley | Cisco Press | Exam Cram  
IBM Press | Que | Prentice Hall | Sams

SAFARI BOOKS ONLINE