Introduction 1.1

Why this book?

If this book had to have a mission statement, we would say that it is designed to help you make the transition from computer *user* to computer *programmer*.[1]

We wrote this book with life scientists in mind. But it is equally appropriate for anyone who needs to slice and dice large, diverse data sets. A few years ago, biologists did not need to know how to program. With the arrival of the Human Genome Project and other -omic technologies, biology has been transformed into an incredibly data-rich science. While the science is moving ahead at a staggering rate, most people have not changed themselves to match. Not everyone needs to know how to program, but for those that desire it, this book will help them catch up quickly.

We have both watched students struggle with trying to analyze mountains of data, and sometimes the struggle has not been because the students lack the ability to tackle the problem. Rather, it is because they frequently lack the *tools* to tackle the problem. For many people, data analysis means ‘using a spreadsheet.’ Sometimes this is all you need, but for many problems a programming solution will be faster, easier, and much more powerful.

This is not a book for dummies or idiots. Conversely, it’s also not for super-geniuses. It’s for ordinary educated people who haven’t needed to program until now. Whether the topic is language, mathematics, or programming, some people learn faster than others. But we all learn to read, write, multiply, and divide. And we can all learn to program. Rest assured, you *can* program. We are happy to be your guides.

Learning to program is a journey. Like other journeys, it takes time and effort. But the rewards are worth every step. Not only will you be learning a new skill that you can apply to your work, you will be seeing the world of data from a completely different perspective. We guarantee you will find this personally enlightening, and we are not exaggerating when we say that your newfound knowledge will empower you more than you can imagine.

Why Unix?

The Unix OS has been around since 1969 and it’s not likely to disappear any time soon. Back then there was no such thing as a graphical user interface (GUI). You typed everything. It may seem archaic to use a keyboard to issue commands today, but it’s much easier to automate keyboard-driven tasks than mouse-driven tasks. There are several variants of Unix (including Linux), though the differences do not matter much. Although you may not have noticed it, Apple has been using Unix as the underlying OS on all of their computers since 2001.[2]

[1] Note that this doesn’t mean you need to grow a beard, start reading science-fiction books, or wear T-shirts bearing unfathomable geeky slogans. Indeed, all of these clichés about programmers should be tossed aside. Programmers are real people … well, most of us are anyway.

[2] If you haven’t noticed it, that’s probably because it is ‘hidden’ behind a very slick-looking GUI. But it’s there nonetheless.

Increasingly, the raw output of biological research exists as *in silico* data, usually in the form of very large text files that can grow to several gigabytes in size. Unix is particularly suited to working with such files and has several powerful (and flexible) commands that can process your data for you. The real strength of learning Unix is that most of these commands can be combined in an almost unlimited fashion. If you can learn just five Unix commands, you will be able to do a lot more than just five things.

Why Perl?

Perl is one of the most popular programming languages, and has a particularly strong following in the bioinformatics community. People sometimes get argumentative about which language is best. There is no single best language for everything. Perl does most things very well, and is a fine programming language to learn. Other equally capable and easy to use languages include Python and Ruby. Once you learn how to program well in one language, adapting to other languages is trivial.

Originally developed in 1987, Perl remains under active development and there is therefore a *lot* of supporting material available to help you learn it.[3] You are very likely to find Perl pre-installed on just about every type of Unix/Linux-based OS, and it is also available for Windows.

Among programming languages, there is often a distinction between those that are *interpreted* (e.g., Perl, Python, Ruby) and those that are *compiled* (e.g., C, C++, Java). People often call interpreted programs *scripts*. It is generally easier to learn programming with a scripting language because you don't have to worry as much about variable types and memory allocation. The downside is that the interpreted programs often run much slower than compiled ones. But let's not get lost in petty details. Scripts are programs, scripting is programming, and computers can solve problems quickly regardless of the language.

About the authors

Keith Bradnam started out his academic career studying ecology. This involved lots of field trips and throwing quadrats around on windy hillsides. He was then lucky enough to be in the right place at the right time to do a Masters degree in bioinformatics (at a time when nobody was very sure what bioinformatics was). From that point onwards he has spent most of his waking life sat at a keyboard (often staring into a Unix terminal). A PhD studying eukaryotic genome evolution followed; this was made easier by the fact that only one genome had been completed at the time he started (this soon changed). After a brief stint working on an *Arabidopsis* genome database he moved to working on the excellent model organism database, WormBase, at the Wellcome Trust Sanger Institute. It was here that he first met Ian Korf and where they bonded over a shared love of Macs, neatly written code, and English puddings. Ian then tried to run away and hide in California at the UC Davis Genome Center, but Keith tracked him down and joined his lab. Apart from doing research, he also gets to look after all the computers in the lab

[3] A good 'first port of call' would be www.perl.org, the official web site of the Perl programming language.

and teach the occasional class or two. However, he would give it all up for the chance to be able to consistently beat Ian at foosball, but that seems unlikely to happen anytime soon. Keith still likes Macs and neatly written code, but now has a much harder job finding English puddings.

As a youth, Ian Korf's favorite classes were sciences and his favorite pastime was computer gaming. At the time, you wouldn't have thought that hacking and writing computer games would be very useful skills for a budding molecular biologist. Certainly nobody ever counseled Ian to do so, especially when he was doing it at 2 a.m.! But apparently the misspent hours of youth can sometimes turn out to be worthwhile investments. Ian's first experience with bioinformatics came as a post-doc at Washington University (St. Louis), where he was a member of the Human Genome Project. He then went 'across the pond' to the Sanger Institute for another post-doc. There he met Keith Bradnam, and found someone who truly understood the role of communication and presentation in science. Ian was somehow able to persuade Keith to join his new lab in Davis, California. This book is but one of their hopefully useful contributions.

Acknowledgments

This book evolved from a course that we both teach to graduate and undergraduate students at UC Davis. We are grateful to the students for their patience with us, as this course has evolved quite a bit since we started teaching it. Their feedback, and their enthusiasm for learning Unix and Perl, have made this book what it is. We also would like to thank Nancy Parmalee for helpful suggestions about the index.

Keith would also like to thank his wife Mel for her tireless support and understanding throughout the long book-writing process. He would also like to express profound gratitude to the wonders of caffeine, the relaxation afforded by his iTunes music library, and to the entire nation of Belgium.

Ian would like to thank all his students, past, present, and future. May your adventures take you to lands unimagined, and your skills see you safely home.

This book was written using Apple's excellent 'Pages' word-processing software, with extensive use of Dropbox software by Dropbox Inc. to make the process of collaborative writing a joy. Code examples were written using TextMate by MacroMates Ltd and TextWrangler by Bare Bones Software.

How to use this book 1.2

Or rather, how *not* to use this book

Organization

This book is divided into seven parts (you are currently reading Part 1). You may be impatient to start programming with Perl, but if you don't know any Unix we suggest that you start with Part 3, which will teach you the basics of Unix. When you finish that, you can optionally jump ahead to Part 5, which covers some advanced Unix topics. Or you might just want to proceed to Part 4, which covers all of the fundamentals of Perl. The choice is yours. Of course, if you don't yet have Unix and Perl installed on your computer, then you might want to start with Part 2, which covers how you can get Unix and Perl for your PC.

If you've never programmed, we hope that after learning the 'essential' Perl of Part 4, you will be able to write many fantastic and powerful scripts. More importantly, we hope that you will be able to write scripts that are actually *useful*. For this part of the book, we've tried, where possible, to only ever introduce one new concept at a time. Hopefully this will prevent you from being overloaded with too many new concepts at once. This also keeps chapters short and, mostly, self-contained. For a few topics that have increased complexity, we use two or more chapters to cover all aspects of that topic.

We have strived to make sure there are lots of examples. These are all scripts that we encourage you to copy and try yourself. However, you may still gain much understanding just from reading them. In addition to the examples, Part 4 of this book also features a number of problems at the end of most chapters.[4] You are strongly encouraged to tackle the problems. Ultimately, this it the best way to learn Perl (or any programming language). For each problem we provide a solution,[5] but be aware that one of the famous mottos associated with Perl is:

TMTOWTDI[6] – There's more than one way to do it

We have hopefully provided solutions that are easily understandable, but if you want to solve each problem in a different way then that is great.

The topics covered in Part 4 might be all you ever need to know in order to solve many different problems. However, we go further into the more advanced aspects of Perl in Part 6. The distinction between 'essential' and 'advanced' is somewhat arbitrary. If you finish Part 4 then you should at least have a look at Part 6.

Part 7 covers many different subjects that are not unique to Perl. In general, this is the section that focuses on 'good programming practices.' Most subjects in this part are relevant to many programming languages, though we also include two sections on how to fix broken Perl scripts.

Finally, we should note that we do not cover every aspect of Unix and Perl. The world of Unix is especially vast, and several books would be needed in order to cover

[4] We include some problems in the Unix section too, but not as many.
[5] Included in an appendix.
[6] Some people pronounce this 'Tim Toady.'

the myriad number of Unix commands you could learn about. Likewise, we do not cover every feature or function available in Perl. However, we strongly feel that this book covers all of the basics (and much more besides). Readers are therefore encouraged to use this book as a launch pad for a journey into a much wider world of programming. If you develop a hunger for learning about new Unix commands, Perl functions, and even new programming languages, then dare to venture beyond the confines of this book. You will be rewarded!

Style conventions

Each chapter has a main heading and a subheading. The subheadings are one area where we have tried to inflict our pun-tastic sense of humor on you.[7] We will often include both Unix and Perl examples, which you should attempt to follow. The Unix examples will include simple instructions of Unix commands that you should type, whereas the Perl examples will contain complete scripts, accompanied by line numbering. E.g.:

Example 1.2.1

```
1.   #!/usr/bin/perl
2.   print "The shortest script in the world?\n";
```

The line numbers are just there so we can refer to them in the text. You are not meant to type the line numbers! Following just about every example will be a section that tries to explain what the point of the example was. For the Unix examples, this will be a section titled *Explanation*, but for the Perl scripts it will be a line-by-line breakdown of how the script is working. E.g.:

Understanding the script

Line 2 contains a simple `print` statement.

In addition to having worked-through examples, we will also set problems that you should try to solve. Where appropriate, answers will be provided, but we encourage you to try solving the problems without looking at the solution.

Hopefully you will have noticed that we use a fixed-width font for writing any Perl or Unix code. This will be done for complete scripts and even when we mention a single Unix command or Perl function within a sentence. E.g., we might mention that the Unix command `sed` shares similarities with Perl's substitution operator (`s//`).

Sometimes we will show fragments of Perl scripts, just to illustrate a point or to demonstrate the syntax of a command. We do not include line numbers for these examples, and they are not intended to be run as complete scripts. E.g.:

```
my @array_A = @array_B; # copying an array
```

[7] Footnotes, like this one, are another place where you might find occasional diversionary comments on matters which may not be entirely related to Unix and Perl. E.g., did you know that there are no words in the English language that rhyme with the word 'orange'?

Occasionally, we will want to shout something at you because it is *so* important, and the world will cease to exist if you fail to understand the critical point we are making. E.g.:

The world will cease to exist if you fail to understand the critical point we are making!

Any time you see something written in this style, you should probably re-read it several times and remember that we will be not-inconsiderably displeased if you fail to remember our advice!