Throughout the book, graphical elements help to highlight certain sections. Icons in the margin point out passages of the book which may be of special interest or importance:


A tip to make your work easier

A potential pitfall that might trip you up


A pitfall that might cause widespread failure


Sections that vary for Windows users. Refer to Appendix 1 or a marginal note


Sections that vary for Linux users, including Linux via Virtual Box within Windows

Text is also formatted to indicate its usage:

System items include files and folders, program names, and menu commands

`Code characters` indicate programming language and terminal commands. The background color indicates whether they are from a `terminal window` or part of a script in a `text document`.

We have customized the `Courier` typeface used to represent code characters to more clearly distinguish between the number `1`, lowercase `l`, and uppercase `I`, as well as between the number `0` and the capital letter `O`. At places in the text, a tab character is represented by the symbol ▵. Further information can be found at the companion site `http://practicalcomputing.org/`.

TEXT FILES

Chapter 1 Getting Set Up

Chapter 2 Regular Expressions: Powerful Search & Replace

Chapter 3 Exploring the Flexibility of Regular Expressions

Chapter 1

GETTING SET UP

Before setting out to learn new computing skills, you will need to take care of a couple of things. Here we describe how to prepare your computer so that you can easily follow along with the examples and apply the new tools we cover. We also explore some fundamental concepts about text files.

An introduction to text manipulation

A variety of seemingly complex and disparate computational challenges routinely faced by biologists can be reformulated as simple text manipulations that can be addressed with a common set of tools. These tasks include aggregating data from several sources that are each formatted differently, or reformatting text files from an instrument or database so they can be further processed by another program. Learning how to automate such tasks can save time and allow you to broaden the scale of your work. There is a good chance that the skill set you build addressing one challenge with general text manipulation skills will be directly applicable to an entirely different challenge in the future. Focusing on general approaches to text handling, therefore, is often a better use of your time than learning the idiosyncrasies of how to get a particular program to accomplish one specific task.

Plain text files are the common currency of many programming environments and their accompanying data. Even for software packages or instruments which store data in special proprietary file formats, there is usually an option to export or import data as plain text files. This provides a way to do things with data beyond what the people writing the software might have intended. You can, for instance, export data from a program frequently used in one discipline and then import it into an analysis program used in a different discipline. The trick to this, though, is that the text file generated by one program is rarely suitable for import as-is into another program; usually it must first be modified.

Learning how to do this will empower you to perform fundamentally new analyses and help your research break new ground, and is a primary goal of this book.

What are text files?

All computer files consist of a linear sequence of binary numbers. Text files are just a special type of binary file where those numbers correspond to human-readable characters such as digits, letters, and punctuation, and white space such as spaces, tabs, and line endings.

REPRESENTING TEXT WITH NUMBERS The correspondence of particular numbers to particular characters follows agreed-upon conventions, the most common of which is the American Standard Code for Information Interchange (ASCII). For instance, in ASCII the number 65 (`0100 0001` in binary) represents the uppercase letter `A`, and the number 49 (`0011 0001` in binary) represents the digit `1`. Notice that in this last example, the digit `1` is not stored as a direct binary representation of the number 1 (that is, `0000 0001`). Nor is a multidigit number such as `12` stored as a direct binary representation of the number 12 (that is, `0000 1100`); instead, it is stored as two sequential codes for the digits `1` and `2`. Text files are inherently more human-friendly than computer-friendly—a person can read them as-is, but a computer must interpret them to use the data for calculations or other analyses.

Appendix 6 shows the complete set of characters encoded by the ASCII standard. An extended character set, including accents and special symbols, is supported by another standard called Unicode, which offers encodings such as UTF-8 and UTF-16. International readers are especially likely to encounter Unicode.

Although all files are inherently binary in nature, a file is usually referred to as being either a **binary file** (a file that doesn't simply encode a series of text characters) or a **text file** (a binary file that does encode such a series). There are many sequences of binary information that don't represent text and can't even be displayed as such. It is more compact, and faster for the computer, to store numbers and other data as direct binary encodings, rather than as the letters and digits describing their values. These performance considerations were more important in the past on slower computers with less memory.

For most biological data, a text representation of data works perfectly well, and is in fact desirable. Why? It comes down to transparency and portability trumping size and speed. Transparency means that the contents of text files are readily accessed. With a binary file, you need a map of how the data are stored to be able to extract them; with a text file, you don't. It can be very difficult to peer into a binary data file and see what's there without thorough documentation of the file structure. A JPEG file, for example, contains compressed binary image data. Try opening it in a text editor, and you'll see that it looks like gibberish. Portability becomes an issue if the program that generated certain files stops being supported or distributed. It is not uncommon to see an ancient computer limping along in the

corner of a lab, just to be able to run a 15-year-old program that is the only means of extracting data from an obsolete file format.

In contrast, the organization of data within a well-conceived text file should be entirely self evident to anyone who opens the file in any text editor. Even if the program that generated a particular type of text file goes away entirely, the ability to just look inside such files and see what data they contain will allow you to rearrange, extract, and continue to utilize those data. So storing data in a well-organized text file greatly extends the lifetime of the data and increases the number of things you can do with them.

The organization of data within a text file

The most common way to organize data in a text file is as **character-delimited text**. The layout is two-dimensional, with columns and lines. The fields within each line are separated by a particular character—the delimiter—which is usually a comma, space, or tab. If the character used to delimit the text occurs within one of the fields, quotation marks are usually placed around the entire field. This indicates that all the characters within are part of the same datum, and that if the delimiter character occurs within the quotes, it does not imply a separation of values as it normally would. The first line in character-delimited text files often consists of a series of descriptions of the columns, delimited by the same character as the data themselves. This description line is called the **header**.

Character-delimited text is an excellent way to handle two-dimensional data, such as a series of measurements taken through time or across multiple samples. Many data loggers and instruments can be set to store and export their data in this way. All spreadsheet programs can export delimited text, usually offering you options for which character to use for separation. However, character delimitation is not the best option for all types of data. If the data are hierarchical in nature, for instance with varying degrees of nestedness and a different number of nested elements at each level, it is a very inefficient way to store information. A phylogenetic tree is an example of this category of data. Character delimitation also breaks down if the fields are very large and vary in length from sample to sample, as is the case with molecular sequence data. Large data values are more conveniently stored across several lines, rather than in a single column in a table.

For these and other reasons, a wide array of formats is available for organizing different types of biological data within text files, often with many formats for the exact same type of data. For example, there are FASTA, GenBank, and NRBF formats for molecular sequence data, and SDTS, Google Earth, and GRASS formats for spatial data.

We will show you how to extract data from one type of text file format, reorganize it, and store it in another type. Later we'll guide you through some of the steps involved in identifying why a file that is supposed to be understood by a particular program isn't, and then using these same reorganizing skills to fix the problem.

Text editors

Given the importance of text files for science, one of the applications that you'll spend the most time interacting with will be a text editor. Although you might be tempted to start with a word processor you are already familiar with, word processors don't actually save plain text. Their files include formatting and layout information in addition to the text to be displayed. Not only is this formatting information unnecessary, but it will corrupt your files by filling them up with unexpected characters. You will almost never use a word processor for any stage of manipulating text data or scripts. Instead, there is a wide variety of text editor programs that focus exclusively on the raw characters found within a text file. These editors show you exactly what is in the file, character by character, and provide a range of useful tools that make it easier to view and edit these characters.

Installing TextWrangler

Although OS X comes with a text editor (appropriately called TextEdit, and located within your Applications folder) we do not recommend that you use it for the tasks we describe in this book. This is because it is actually a hybrid between a word processor and a text editor, sometimes leading to confusion about when you are seeing formatted text versus raw text, and it isn't optimized for displaying and editing data and software. Instead, we recommend that you download TextWrangler. This is the free version of BBEdit, and includes most of that editor's important features. There are many great text editors available, but TextWrangler is very powerful and distributed free of charge. All the editing in this book can be reproduced using this program.

OTHER TEXT EDITORS TextWrangler is used for the examples throughout this book. If you are using Windows, we suggest you use Notepad++ as an equivalent. For Linux, we recommend either gedit, or the popular text editor jEdit (consult Appendix 1 for details). There are many text editors, and each has its camp of advocates. Most commercial products are reasonably priced. If you also want to try other editors, some popular alternatives for OS X include BBEdit and TextMate, each of which has its strengths, or Emacs, which is popular among programmers. Because programmers spend so much time doing repetitive tasks in text editing programs, an extensive feature set has been built into many text editors to streamline operations and check errors as you go. These features include coloring text based on the context, auto-completing the names of items in your scripts, filling in templates with commonly used program snippets, and reformatting your programs for clarity.

You can download TextWrangler from the BareBones Software Web site, `www.barebones.com/products/textwrangler`. We used version 3.0, and there may be minor differences in other versions. TextWrangler shows multiple open documents in a single window, organized in a drawer to the right of the editing window that you can use to toggle between them. After you install TextWrangler, it won't necessarily open all the appropriate files by default when you double-click them. You may need to drop some files onto the program's icon or use the Open dialog box within TextWrangler to access them.

Optimizing text appearance within a text editor

Although you will be dealing mostly with text files that contain no information on formatting, you still have control over how the text is displayed on the screen through the text editor preferences. You can, for instance, set the display font. For data and programs, it is usually desirable to use a fixed-width font (also called a fixed-space or monospaced typeface), such as Courier. The advantage of these fixed-space fonts is that data and program code will line up better from line to line since each character has the same width. You can do a quick test to see if your typeface is monospaced by typing the letters iiii and OOOO on consecutive lines:

Proportionally spaced	Fixed-space
iiii	`iiii`
OOOO	`OOOO`

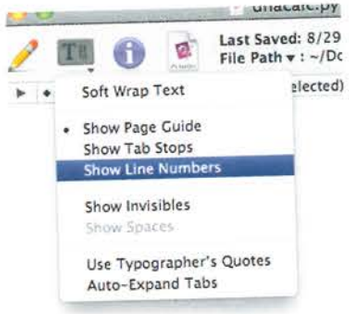
If the left and right edges of the sets of characters line up, you probably have a monospaced typeface.

Most text editors also provide the handy feature of displaying line numbers to the left of the text. This is helpful for navigating large files and going straight to problems in the file that were identified during testing by other programs. To enable line numbers in TextWrangler, open an existing or blank document, click on the Text Options icon at the top of the window, and select Show Line Numbers in the pull-down menu.

Remember, these display settings do not alter the text file itself—they are just editor settings that control how file contents are displayed. You can have different display settings for different files, but these are not stored in the file as with word processors. The text editor is separately keeping track of your preferences for each file.

Line endings

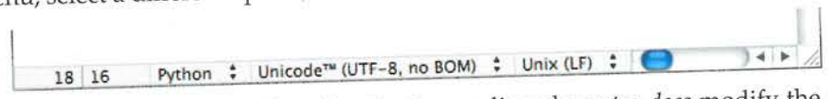
The characters used to designate the ending of a line—that is, the symbol inserted when you hit the return key—were never standardized across different operating systems. This may seem like an esoteric technical issue, but it is one of the most common causes of problems when transferring text files from one computer to another, or even between programs on the same computer. It is likely that you have



come across this problem before, such as when opening a result file from a remote computer and finding that all the contents are shown on a single line. It is also one of the most common reasons that a program will not accept text data that appear to be correctly organized. Many programs that can't handle different types of line endings do not give you an error telling you this—they just crash. If you can't get something to run, it is a good idea to first check the line endings of your input files.

There are two line ending characters, **carriage return** (abbreviated CR or `\r`) and **line feed** (abbreviated LF or `\n`). Unix systems, including OS X, use a line feed, some older Macintosh programs use a carriage return, and Microsoft Windows uses a carriage return followed by a line feed! When it is within your control to decide which line ending to use, or when you are making a first stab and don't know what to expect, go with a line feed. It will generally give you the fewest problems across systems.

You can check which line endings are used within a file by opening it in TextWrangler and taking a look at the status bar at the bottom of the window. Next to the horizontal scroll bar there is a drop-down menu that shows the current line ending character for the file. To change this character, just click the drop-down menu, select a different option, and save the file.



Unlike cosmetic changes, changing the line ending character *does* modify the file by altering which characters are recorded within the file itself.

The example files

You will be working with a variety of example files throughout this book. These are available for download as a zip file at `http://www.practicalcomputing.org`.

Installing the example files

After downloading the zip file, place it in your home folder. This is the folder where your Documents and Pictures folders are stored by default; it is also the folder that typically opens if you create a new Finder window on OS X.[1] Double-click the zip file to extract its contents; this will generate a new folder called pcfb. The pcfb folder has three folders within it. The first folder, examples, has the data files you will need to follow along with the book. The second, scripts, has programs and scripts you will use and modify. The third, sandbox, is an empty folder

[1] If you don't know what your home directory is, see the explanation accompanying Figure 4.2. For Windows and Linux users, put the folder in the directory named with your username, and see Appendix 1 for more information on the file structure.

that provides a convenient place to work with the files you will generate. Many of the examples will assume that these folders are located in the specific location described above—so if you put it somewhere else, you will need to adapt the examples accordingly.

Exploring the example files

Go ahead and use a text editor to explore some of the files in the examples folder you installed earlier. First, from within TextWrangler (or the alternative text editor for your computer) open ctd.txt. This file contains columns of comma-delimited measurements of salinity, temperature, oxygen concentration, and various other measurements at different depths off the coast of California. Now, open ctd.rtf from within TextWrangler. This file contains the exact same data but has been edited in a word processor and now contains hidden formatting information (font, font size, and a bold header). In addition to the data, you can see all kinds of characters that explain to a word processor how to display the text, but don't represent data. This illustrates why you want to stick to a raw text editor like TextWrangler when handling data: the added formatting information can confuse the interpretation of the file by other programs. With a raw text editor, what you see is what you get—all the contents of the text file are shown and you know exactly what you are dealing with.

Now, in your computer's file browser (not the Open dialog box in TextWrangler), find the same ctd.rtf file and double-click it. This will open it in a word processor, probably the hybrid program TextEdit that we mentioned earlier. You will see that none of the formatting characters are shown when using a word processor, just the formatted data. It is difficult to know what the actual contents of the file are, and consequently how they would be interpreted by other programs. Though it is often tempting to open raw data files in a word processor and use different fonts, colors, and highlights to organize different types of information within the file, this peek inside should begin to give you a sense of the perils of following such an approach. It is far clearer, and ultimately more flexible, to use well-defined strategies for organizing and explicitly annotating data (e.g., with column headers) in a plain text file.

SUMMARY

You have learned:

- The nature of text files
- Broad considerations regarding the organization of data within text files
- What text editors are, and how to install the TextWrangler editor
- The different line endings that are used in text files
- How to install the example files used throughout the book