

# The Pragmatic Programmer



from journeyman  
to master

Andrew Hunt  
David Thomas

## What others in the trenches say about *The Pragmatic Programmer* . . .

“The cool thing about this book is that it’s great for keeping the programming process fresh. *[The book]* helps you to continue to grow and clearly comes from people who have been there.”

- **Kent Beck**, author of *Extreme Programming Explained: Embrace Change*

“I found this book to be a great mix of solid advice and wonderful analogies!”

- **Martin Fowler**, author of *Refactoring* and *UML Distilled*

“I would buy a copy, read it twice, then tell all my colleagues to run out and grab a copy. This is a book I would never loan because I would worry about it being lost.”

- **Kevin Ruland**, Management Science, MSG-Logistics

“The wisdom and practical experience of the authors is obvious. The topics presented are relevant and useful. . . . By far its greatest strength for me has been the outstanding analogies—tracer bullets, broken windows, and the fabulous helicopter-based explanation of the need for orthogonality, especially in a crisis situation. I have little doubt that this book will eventually become an excellent source of useful information for journeymen programmers and expert mentors alike.”

- **John Lakos**, author of *Large-Scale C++ Software Design*

### Using Unix Tools Under Windows

We love the availability of high-quality Unix tools under Windows, and use them daily. However, be aware that there are integration issues. Unlike their MS-DOS counterparts, these utilities are sensitive to the case of filenames, so `ls a*.bat` won't find `AUTOEXEC.BAT`. You may also come across problems with filenames containing spaces, and with differences in path separators. Finally, there are interesting problems when running MS-DOS programs that expect MS-DOS-style arguments under the Unix shells. For example, the Java utilities from JavaSoft use a colon as their `CLASSPATH` separator under Unix, but use a semicolon under MS-DOS. As a result, a Bash or ksh script that runs on a Unix box will run identically under Windows, but the command line it passes to Java will be interpreted incorrectly.

Alternatively, David Korn (of Korn shell fame) has put together a package called UWIN. This has the same aims as the Cygwin distribution—it is a Unix development environment under Windows. UWIN comes with a version of the Korn shell. Commercial versions are available from Global Technologies, Ltd. [URL 30]. In addition, AT&T allows free downloading of the package for evaluation and academic use. Again, read their license before using.

Finally, Tom Christiansen is (at the time of writing) putting together *Perl Power Tools*, an attempt to implement all the familiar Unix utilities portably, in Perl [URL 32].

### Related sections include:

- *Ubiquitous Automation*, page 230

### Challenges






- Are there things that you're currently doing manually in a GUI? Do you ever pass instructions to colleagues that involve a number of individual "click this button," "select this item" steps? Could these be automated?
- Whenever you move to a new environment, make a point of finding out what shells are available. See if you can bring your current shell with you.
- Investigate alternatives to your current shell. If you come across a problem your shell can't address, see if an alternative shell would cope better.

## Power Editing

We've talked before about tools being an extension of your hand. Well, this applies to editors more than to any other software tool. You need to be able to manipulate text as effortlessly as possible, because text is the basic raw material of programming. Let's look at some common features and functions that help you get the most from your editing environment.

### One Editor

We think it is better to know one editor very well, and use it for all editing tasks: code, documentation, memos, system administration, and so on. Without a single editor, you face a potential modern day Babel of confusion. You may have to use the built-in editor in each language's IDE for coding, and an all-in-one office product for documentation, and maybe a different built-in editor for sending e-mail. Even the keystrokes you use to edit command lines in the shell may be different.<sup>4</sup> It is difficult to be proficient in any of these environments if you have a different set of editing conventions and commands in each.

You need to be proficient. Simply typing linearly and using a mouse to cut and paste is not enough. You just can't be as effective that way as you can with a powerful editor under your fingers. Typing  or  ten times to move the cursor left to the beginning of a line isn't as efficient as typing a single key such as , , or .

#### TIP 22

#### Use a Single Editor Well

Choose an editor, know it thoroughly, and use it for all editing tasks. If you use a single editor (or set of keybindings) across all text editing activities, you don't have to stop and think to accomplish text manipulation: the necessary keystrokes will be a reflex. The editor will be

---

4. Ideally, the shell you use should have keybindings that match the ones used by your editor. Bash, for instance, supports both vi and emacs keybindings.

an extension of your hand; the keys will sing as they slice their way through text and thought. That's our goal.

Make sure that the editor you choose is available on all platforms you use. Emacs, vi, CRISP, Brief, and others are available across multiple platforms, often in both GUI and non-GUI (text screen) versions.

## Editor Features

Beyond whatever features you find particularly useful and comfortable, here are some basic abilities that we think every decent editor should have. If your editor falls short in any of these areas, then this may be the time to consider moving on to a more advanced one.

- **Configurable.** All aspects of the editor should be configurable to your preferences, including fonts, colors, window sizes, and key-stroke bindings (which keys perform what commands). Using only keystrokes for common editing operations is more efficient than mouse or menu-driven commands, because your hands never leave the keyboard.
- **Extensible.** An editor shouldn't be obsolete just because a new programming language comes out. It should be able to integrate with whatever compiler environment you are using. You should be able to "teach" it the nuances of any new language or text format (XML, HTML version 9, and so on).
- **Programmable.** You should be able to program the editor to perform complex, multistep tasks. This can be done with macros or with a built-in scripting programming language (Emacs uses a variant of Lisp, for instance).

In addition, many editors support features that are specific to a particular programming language, such as:

- Syntax highlighting
- Auto-completion
- Auto-indentation
- Initial code or document boilerplate
- Tie-in to help systems
- IDE-like features (compile, debug, and so on)

Figure 3.1. Sorting lines in an editor

```

import java.util.Vector;   emacs: M-x sort-lines   import java.awt.*;
import java.util.Stack;    ↗                      import java.net.URL;
import java.net.URL;       ↘                      import java.util.Stack;
import java.awt.*;         vi: :.,+3!sort          import java.util.Vector;

```

A feature such as syntax highlighting may sound like a frivolous extra, but in reality it can be very useful and enhance your productivity. Once you get used to seeing keywords appear in a different color or font, a mistyped keyword that *doesn't* appear that way jumps out at you long before you fire up the compiler.

Having the ability to compile and navigate directly to errors within the editor environment is very handy on big projects. Emacs in particular is adept at this style of interaction.

## Productivity

A surprising number of people we've met use the Windows notepad utility to edit their source code. This is like using a teaspoon as a shovel—simply typing and using basic mouse-based cut and paste is not enough.

What sort of things will you need to do that *can't* be done in this way?

Well, there's cursor movement, to start with. Single keystrokes that move you in units of words, lines, blocks, or functions are far more efficient than repeatedly typing a keystroke that moves you character by character or line by line.

Or suppose you are writing Java code. You like to keep your `import` statements in alphabetical order, and someone else has checked in a few files that don't adhere to this standard (this may sound extreme, but on a large project it can save you a lot of time scanning through a long list of `import` statements). You'd like to go quickly through a few files and sort a small section of them. In editors such as `vi` and Emacs you can do this easily (see Figure 3.1). Try *that* in notepad.

Some editors can help streamline common operations. For instance, when you create a new file in a particular language, the editor can supply a template for you. It might include:

- Name of the class or module filled in (derived from the filename)
- Your name and/or copyright statements
- Skeletons for constructs in that language (constructor and destructor declarations, for example)

Another useful feature is auto-indenting. Rather than having to indent manually (by using space or tab), the editor automatically indents for you at the appropriate time (after typing an open brace, for example). The nice part about this feature is that you can use the editor to provide a consistent indentation style for your project.<sup>5</sup>

## Where to Go from Here

This sort of advice is particularly hard to write because virtually every reader is at a different level of comfort and expertise with the editor(s) they are currently using. So, to summarize, and to provide some guidance on where to go next, find yourself in the left-hand column of the chart, and look at the right-hand column to see what we think you should do.

### ***If this sounds like you...***

### ***Then think about...***

*I use only basic features of many different editors.*

Pick a powerful editor and learn it well.

*I have a favorite editor, but I don't use all of its features.*

Learn them. Cut down the number of keystrokes you need to type.

*I have a favorite editor and use it where possible.*

Try to expand and use it for more tasks than you do already.

*I think you are nuts. Notepad is the best editor ever made.*

As long as you are happy and productive, go for it! But if you find yourself subject to “editor envy,” you may need to reevaluate your position.

---

5. The Linux kernel is developed this way. Here you have geographically dispersed developers, many working on the same pieces of code. There is a published list of settings (in this case, for Emacs) that describes the required indentation style.

## What Editors Are Available?

Having recommended that you master a decent editor, which one do we recommend? Well, we're going to duck that question; your choice of editor is a personal one (some would even say a religious one!). However, in Appendix A, page 266, we list a number of popular editors and where to get them.

### Challenges

- Some editors use full-blown languages for customization and scripting. Emacs, for example, uses Lisp. As one of the new languages you are going to learn this year, learn the language your editor uses. For anything you find yourself doing repeatedly, develop a set of macros (or equivalent) to handle it.
- Do you know everything your editor is capable of doing? Try to stump your colleagues who use the same editor. Try to accomplish any given editing task in as few keystrokes as possible.

## Source Code Control

*Progress, far from consisting in change, depends on retentiveness. Those who cannot remember the past are condemned to repeat it.*

► **George Santayana, *Life of Reason***

One of the important things we look for in a user interface is the UNDO key—a single button that forgives us our mistakes. It's even better if the environment supports multiple levels of undo and redo, so you can go back and recover from something that happened a couple of minutes ago. But what if the mistake happened last week, and you've turned your computer on and off ten times since then? Well, that's one of the many benefits of using a source code control system: it's a giant UNDO key—a project-wide time machine that can return you to those halcyon days of last week, when the code actually compiled and ran.

Source code control systems, or the more widely scoped *configuration management* systems, keep track of every change you make in your source code and documentation. The better ones can keep track of