0 Preface

This book is about porting software between UNIX platforms, the process of taking a software package in source form and installing it on your machine. This doesn't sound like a big deal at first, but there's more to it than meets the eye: you need to know how to get the software, how to unpack what you get, how to modify the package so that it will compile on your system, how to compile and install the software on your system, and how to deal with problems if they crop up.

Nevertheless, it doesn't involve anything that hasn't already been done to death in hundreds of well-written books: you can find out about getting software from the Internet in *The Whole Internet User's Guide and Catalog*, by Ed Krol. Unpacking software is basically a matter of using standard tools described in dozens of good introductory textbooks. Compiling programs is so simple that most C textbooks deal with it in passing. Installation is just a matter of copying software to where you want it. Programming is the meat of lots of books on UNIX programming, for example *Advanced Programming in the UNIX environment* by Richard Stevens,

So why yet another book?

Most textbooks give you an idealized view of programming: "This is the way to do it" ("and it works"). They pay little attention to the ways things can go wrong. UNIX is famed for cryptic or misleading error messages, but not many books go into the details of why they appear or what they really mean. Even experienced programmers frequently give up when trying to port software. The probable advantage of completing the port just isn't worth effort that it takes. In this book, I'd like to reduce that effort.

If you take all the books I just mentioned, you'll have to find about 3 feet of shelf space to hold them. They're all good, but they contain stuff that you don't really want to know about right now (in fact, you're probably not sure if you ever want to know all of it). Maybe you have this pressing requirement to get this debugger package, or maybe you finally want to get the latest version of *nethack* up and running, complete with X11 support, and the last thing you want to do on the way is go through those three feet of paper.

That's where this book comes in. It covers all issues of porting, from finding the software through porting and testing up to the final installation, in the sequence in which you perform them. It goes into a lot of detail comparing the features of many different UNIX systems, and offers suggestions about how to emulate features not available on the platform to which you are porting. It views the problems from a practical rather than from a theoretical perspective. You probably won't know any more after reading it than you would after reading the in-depth books, but I hope that you'll find the approach more related to your immediate problems.

Audience

This book is intended for anybody who has to take other people's software and compile it on a UNIX platform. It should be of particular interest to you if you're:

- A software developer porting software to a new platform.
- A system administrator collecting software products for your system.
- A computer hobbyist collecting software off the Internet.

Whatever your interest, I expect that you'll know UNIX basics. If you're a real newcomer, you might like to refer to *Learning the UNIX Operating System*, by Grace Todino, John Strang and Jerry Peek. In addition, *UNIX in a Nutshell*, available in BSD and System V flavours, includes a lot of reference material which I have not repeated in this book.

The less you already know, the more use this book is going to be to you, of course. Nevertheless, even if you're an experienced programmer, you should find a number of tricks to make life easier.

Organization

One of the big problems in porting software is that you need to know everything first. While writing this book I had quite a problem deciding the order in which to present the material. In the end, I took a two-pronged approach, and divided this book into two major parts:

1. In the first part, we'll look at the stages through which a typical port passes: getting the software, extracting the source archives, configuring the package, compiling the software, testing the results, and installing the completed package.
2. In the second part, we'll take a look at the differences between different flavours of UNIX, how they can make life hard for you, and how we can solve the problems.

Operating System Versions

Nearly everything in this book is related to one version or another of UNIX,* and a lot of the text only makes sense in a UNIX context. Nevertheless, it should be of some use to users of other operating systems that use the C programming language and UNIX tools such as *make*.

As in any book about UNIX, it's difficult to give complete coverage to all flavours. The examples in this book were made with six different hardware/software platforms:

* UNIX is, of course, a registered trademark of its current owner. In this context, I am referring to any operating system that presents a UNIX-like interface to the user and the programmer.

- SCO XENIX/386 on an Intel 386 architecture (version 2.3.2).
- UNIX System V.3 on an Intel 386 architecture (Interactive UNIX/386 version 2.2).
- UNIX System V.4.2 on an Intel 386 architecture (Consensys V4.2).
- BSD on an Intel 386 architecture (BSD/386* 1.1 and FreeBSD).
- SunOS on a Sparc architecture (SunOS 4.1.3).
- IRIX 5.3 on an SGI Indy Workstation (mainly System V.4).

This looks like a strong bias towards Intel architectures. However, most problems are more related to the software platform than the hardware platform. The Intel platform is unique in offering almost every flavour of UNIX that is currently available, and it's easier to compare them if the hardware is invariant. I believe these examples to be representative of what you might find on other hardware.

The big difference in UNIX flavours is certainly between UNIX System V.3 and BSD, while System V.4 represents the logical sum of both of them. At a more detailled level, every system has its own peculiarities: there is hardly a system available which doesn't have its own quirks. These quirks turn out to be the biggest problem that you will have to fight when porting software. Even software that ported just fine on the previous release of your operating system may suddenly turn into an error message generator.

Conventions used in this book

This book uses the following conventions:

Bold is used for the names of keys on the keyboard. We'll see more about this in the next section.

Italic is used for the names of UNIX utilities, directories and filenames, and to emphasize new terms and concepts when they are first introduced.

`Constant Width` is used in examples to show the contents of files, the output from commands, program variables, actual values of keywords, for the names of *Usenet* newsgroups, and in the text to represent commands.

`Constant Italic` is used in examples to show variables for which context-specific substitutions should be made. For example, the variable *`filename`* would be replaced by an actual filename. In addition it is used for comments in code examples.

`Constant Bold` is used in examples to show commands or text that would be typed in literally by the user.

Most examples assume the use of the Bourne shell or one of its descendents such as the Korn Shell, or the Free Software Foundation's *bash*. Normally the prompt will be shown as the default `$`, unless it is an operation that requires the superuser, in which case it will be shown as `#`. When continuation lines are used, the prompt will be the standard `>`. In cases where the command wouldn't work with the C shell, I present an alternative. In the C shell examples, the prompt is the default `%`.

* Later versions of this operating system are called BSD/OS.

I have tried to make the examples in this book as close to practice as possible, and most are from real-life sources. A book is not a monitor, however, and displays that look acceptable (well, recognizable) on a monitor can sometimes look really bad in print. In particular, the utilities used in porting sometimes print out "lines" of several hundred characters. I have tried to modify such output in the examples so that it fits on the page. For similar reasons, I have modified the line breaks in some literally quoted texts, and have occasionally squeezed things like long directory listings.

Describing the keyboard

It's surprising how many confusing terms exist to describe individual keys on the keyboard. My favourite is the *any* key (`Press any key to continue`). We won't be using the *any* key in this book, but there are a number of other keys whose names need understanding:

- The **Enter** or **Return** key. I'll call this **RETURN**.
- Control characters (characters produced by holding down the **CTRL** key and pressing a normal keyboard key at the same time). These characters are frequently echoed on the screen as a caret (ˆ) followed by the character entered. In keeping with other Nutshell books, I'll write control-D as **CTRL-D**.
- The **ALT** key, which emacs aficionados call a **META** key, works like a second **CTRL** key, but generates a different set of characters. These are sometimes abbreviated by prefixing the character with a tilde (˜) or the characters **A-**. Although these are useful abbreviations, they can be confusing, so I'll spell these out as **CTRL-X** and **ALT-D**, etc.
- **NL** is the *new line* character. In ASCII, it is **CTRL-J**, but UNIX systems generate it when you press the **RETURN** key.
- **CR** is the *carriage return* character, in ASCII **CTRL-M**. Most systems generate it with the **RETURN** key.
- **HT** is the ASCII *horizontal tab* character, **CTRL-I**. Most systems generate it when the *TAB* key is pressed.

Terminology

Any technical book uses jargon and technical terms that are not generally known. I've tried to recognize the ones used in this book and describe them when they occur. Apart from this, I will be particularly pedantic about the way I use the following terms in this book:

program Everybody knows what a program is: a series of instructions to the computer which, when executed, cause a specific action to take place. Source files don't fit this category: a *source program* (a term you won't find again in this book) is really a *program source* (a file that you can, under the correct circumstances, use to create a program). A program may, however, be interpreted, so a shell script may qualify as a program. So may something like an *emacs* macro, whether byte compiled or not (since *emacs* can interpret uncompiled macros directly).

package A package is a collection of software maintained in a source tree. At various stages in the build process, it will include

- *source* files: files that are part of the distribution.
- *auxiliary* files, like configuration information and object files that are not part of the source distribution and will not be installed.
- *installable* files: files that will be used after the build process is complete. These will normally be copied outside the source tree so that the source tree can be removed, if necessary.

Some software does not require any conversion: you can just install the sources straight out of the box. We won't argue whether this counts as a package. It certainly shouldn't give you any porting headaches.

We'll use two other terms as well: *building* and *porting*. It's difficult to come up with a hard-and-fast distinction between the two—we'll discuss the terms in Chapter 1, *Introduction*.

Acknowledgements

Without software developers all over the world, there would be nothing to write about. In particular, the Free Software Foundation and the Computer Sciences Research Group in Berkeley (now defunct) have given rise to an incredible quantity of freely available software. Special thanks go to the reviewers Larry Campbell and Matt Welsh, and particularly to James Cox, Jerry Dunham, and Jörg Micheel for their encouragement and meticulous criticism of what initially was just trying to be a book. Thanks also to Clive King of the University of Aberystwyth for notes on data types and alignment, Steve Hiebert with valuable information about HP-UX, and Henry Spencer and Jeffrey Friedl for help with regular expressions.

Finally, I can't finish this without mentioning Mike Loukides and Andy Oram at O'Reilly and Associates, who gently persuaded me to write a book about porting, rather than just presenting the reader with a brain dump.