Preface

A language is not worth knowing unless it teaches you to think differently.
—Larry Wall (the creator of Perl) and Randal Schwartz

UNIX and C changed the way people used and learned about computers. Even though technology changes fast—sometimes, every month—certain approaches to technology remain unchanged. UNIX is one of them; it has survived the test of time. It's no wonder that theoretical courses on operating systems often use the UNIX system to illustrate key features. Thanks partly to Linux, things should remain this way in the foreseeable future too.

Yet, UNIX is still described by many as "unfriendly" and "unforgiving." Beginners feel overwhelmed by its sheer weight, and even experienced computer professionals feel lost in this mysterious universe. What bothers them is the UNIX "command line" associated with its myriad options and complex syntaxes. On other systems, the mouse does most of the work, so what's wrong with the UNIX way of doing things?

Nothing, absolutely nothing, it's just that perceptions differ. What is unfriendly and unforgiving to one can be quite friendly and smart to another. In fact, UNIX was deliberately designed to appear "unfriendly." It supports a repository of standalone *commands* that can also be used in combination to solve complex text manipulation problems. Add the shell's programming features, and you can at once develop both interactive and noninteractive applications—and even schedule them to run at specific times. This is what this book attempts to explain and advise: *There is a method to this madness.*

The excitement that UNIX generates lies in the fact that many of its powers are hidden. It doesn't offer everything on a platter; it encourages you to create and innovate. Figuring out a command combination or designing a script that does a complex job is a real challenge to the UNIX enthusiast. This is what UNIX is, and it had better remain that way. If you appreciate this, then you are on the right track, and this book is for you.

How This Book Is Different

I made no conscious decision to make this book different from others. Facing a UNIX box was my first encounter with computers, and prolonged periods of struggle with the system have led me to believe that the stumbling blocks to understanding UNIX are

often different from what they are perceived to be. I couldn't wholeheartedly embrace the way people wrote on the subject, and instead conceived my own idea of the "true" UNIX book—a book that people would like to have with them all the time.

Real—life Examples UNIX concepts are simple but they are also abstract, and it's not often obvious why a certain feature is handled in a particular way. The mastery of this operating system requires a clear understanding of these concepts. I have made sure that the key features are explained clearly to reveal both their design considerations and their relevance in the real world. You'll find that most examples of this text refer to real-life situations.

Both a User's and Programmer's Guide There are mainly two categories of UNIX users: those who *use* its native tools and write shell scripts, and others who *develop* tools using the UNIX system call library. The "user" category is served by the first 13 chapters, which is more than adequate for an introductory UNIX course.

The "developer" is a systems programmer who also needs to know how things work, say, how a directory is affected when a file is created or linked. For their benefit, the initial chapters contain special boxes that probe key concepts. This arrangement shouldn't affect the beginner who may quietly ignore these portions. Systems programmers will find a wealth of material in Chapters 17 and 18 (new in this edition) that will help them in designing new tools using the system call library. I haven't seen any book that adequately addresses both user segments, but this book does.

Strong Learning Aids The pedagogical aids are a strong feature of this book, and you'll find nearly 250 instances of such aids in this text. They take on various names, for example, Note, Caution, and Tip. I consider Linux to be an important member of the UNIX family, so I have separately highlighted Linux features using the penguin as identifier.

I don't agree with the approach adopted by many authors of treating each shell in a separate chapter. Instead, I have discussed key concepts using mainly the Bourne shell, which I still think (even in 2004!) clearly reveals how the system interacts with the user. Deviations have been taken care of by separate asides for the C shell, Korn and Bash shells. Because the Bourne shell lacks sophisticated environment-related features, only a single chapter (Chapter 9) doesn't use Bourne as the main shell. I believe this approach is effective and it also saves time and space.

Numerous Questions and Exercises This book features an enormous number of questions that test the reader's knowledge—over 800 of them. More than a third of them are Self-Test questions, and their answers are provided in Appendix H. These questions are all targeted toward beginners who will do well to answer them before moving on to the next chapter.

More rigorous and extensive questioning is reserved for the Exercises section. Some of them pose real challenges, and it may take you some time to solve them. These exercises reinforce and often add to your knowledge of UNIX, so don't ignore them. The answers to these questions are available to adopters of the book at the book's Web site *http://www.mhhe.com/das*. You'll find a lot of additional material on this site that you can use to supplement this book.

What Has Changed In This Edition

While the fundamental UNIX story remains intact, my perceptions of the operating system have changed with time. This second edition provided me with an opportunity to completely overhaul the structure and content of the previous edition. The result is a concise and yet comprehensive book that has these features:

- Increased emphasis on concepts and their application. Implementation details that don't materially contribute to the understanding of concepts have been trimmed or deleted.
- Shell programming is covered with increased depth and with better examples. Advanced programming features of the Korn and Bash shells have been moved to an appendix.
- Updated coverage of security-related features. The Berkeley-based r-utilities have been removed, and coverage of **`telnet`** and **`ftp`** have been pruned. This edition features the Secure Shell and the basics of Public Key Cryptography.
- The pedagogical features—the Note, Tip and Caution asides and the end-of-chapter questions—have been given a major face-lift. The key chapters now have a larger number of questions, and many of them are quite challenging.
- There are special asides occurring in the beginning chapters that explain how a concept works in UNIX. Skip them if you don't intend to program in UNIX.
- C programmers who intend to use the system call library will be benefited by the inclusion of three new chapters:
  - Chapter 16: Program Development Tools
  - Chapter 17: Systems Programming I—Files
  - Chapter 18: Systems Programming II—Process Control
- It's widely felt that system and network administration deserves a separate book of its own. There's a single chapter this time that examines only the general issues.

These changes have resulted in fewer chapters (19 instead of 24) and fewer pages. However, the generic character of the book has been retained; it doesn't focus on any particular flavor of UNIX, but variations found in Solaris and Linux have been highlighted.

Understanding The Organization

This book is *logically* divided into essential and advanced sections. Essential UNIX is confined to the first 13 chapters that culminate with shell programming. Advanced material that includes TCP/IP tools, **`perl`** and systems programming, are covered in the remaining six chapters.

Introducing UNIX Chapter 1 reveals the key UNIX concepts through a simple hands-on session. This is followed by a brief history and presentation of the features of UNIX. Get introduced to the *kernel* and *shell,* who between them, handle the system's workload. Also understand the role played by standards bodies like POSIX and The Open Group in laying the framework for developing portable applications.

Chapter 2 presents the structure of the UNIX command line. It also discusses the techniques of using the **man** command to look up the online documentation. Also learn to use an email program, change your password and see what's going on in the system. Things can and will go wrong so you also need to know how to use the keyboard for corrective action.

Files The *file* is one of the two pillars that support UNIX, and the next four chapters discuss files. Chapter 3 discusses the various types of files you'll find on your system and the commands that handle them. You'll learn to create directories, navigate a directory structure and copy and delete files in a directory. UNIX also offers a host of compression utilities that you need to use to conserve disk space.

Files have attributes (properties) and Chapter 4 presents the major attributes, especially the ones displayed by the **ls -l** command. Be aware that your files and directories are open to attack, so learn to protect them by manipulating their permissions. Use *links* to access a file by multiple names. You'll also forget where you have kept your files, so you need to be familiar with the **find** command.

How productive you eventually are also depends on how well you exploit the features of your editor. Chapters 5 and 6 presents two of the most powerful text editors found in any operating environment: **vi** and **emacs**. You probably will need to know only one of them. A programmer probably uses the editor more than anyone else, so most examples in these two chapters use snippets of program code. Appendix C presents a summary of their features.

The Shell and Process You now need to understand a very important program that is constantly interacting with you—the shell. Chapter 7 presents the interpretive features of the shell including many of its *metacharacters*. Learn to use *wild cards* to match a group of similar filenames with a single pattern. Manipulate the input and output of commands using *redirection* and *pipes*. The shell is also a programming language, so you have to wait until Chapter 13 to understand it completely.

Chapter 8 introduces the *process* as the other pillar of the UNIX system. Processes are similar to files, and processes also have attributes. Understand how the *fork-exec* mechanism is used to create a process. Learn to control processes, move them between foreground and background and also kill them by sending them *signals*.

The UNIX shell provides excellent opportunities to customize your environment (Chapter 9). Understand and manipulate shell variables, create command *aliases* and use the *history* mechanism to recall, edit and re-execute previous commands. Choose a suitable shell that offers all these features and learn to use the initialization scripts to save the changes you've made to the environment.

Filters and Shell Programming The next three chapters deal with *filters*—those special commands in the UNIX tool kit that handle all text manipulation tasks. Chapter 10 presents the simple ones and shows how they are most effective when they are connected to one another. A special examples section features three real-life applications that are handled by these filters working in pipelines.

Chapter 11 discusses two powerful filters—**grep** and **sed**—that, between them, handle all pattern search, edit and replace operations. At this stage, you'll be introduced to *regular expressions,* an elaborate pattern matching mechanism that often make searching and replacement a lot easier.

When you can't handle a text manipulation problem with the other filters, you need to use **awk**. The command makes its appearance as a filter and a programming language in Chapter 12. Knowing **awk** and its standard programming constructs (like the **if**, **for**, and **while** constructs) should prepare you well for shell programming (and **perl**).

Eventually, you'll place all your commands and pipelines in *shell scripts*. Use the programming features of the shell discussed in Chapter 13 to develop both interactive and noninteractive scripts. Learn to design a script whose behavior depends on the *name* by which it is invoked. The three sample scripts featured in the chapter are compulsory reading for a shell programmer.

These thirteen chapters are all that the beginner needs to know initially. The next six chapters present some of the advanced features of UNIX that also include a chapter on system administration.

Advanced Topics Though networking wasn't part of the UNIX scheme of things, TCP/IP was first ported to UNIX. Chapter 14 covers the networking tools. Get introduced to *hostnames, IP addresses*, and *domain names* and how they are used by the Internet protocols (like HTTP). The chapter also introduces you to Public Key Cryptography and the Secure Shell.

We encounter **perl** in Chapter 15 as the most powerful filter and scripting language in the UNIX world. Most UNIX concepts are embedded in the design of **perl**, the reason why many UNIX users can't do without it. Even though we can't do justice to **perl** in a single chapter, Chapter 15 represents a useful beginning.

The next three chapters are directly or indirectly related to C programming. Chapter 16 presents the program development tools. Use the **make** utility and a powerful debugger for managing and debugging programs. Also, learn to maintain multiple versions of a program using SCCS and RCS.

Chapter 17 is the first of two chapters that feature the use of *system calls* in the C programming environment. This chapter discusses the system calls related to files and I/O. Write programs that perform directory-oriented functions like listing files. Also learn to fetch and manipulate file attributes stored in the inode.

Chapter 18 discusses the system calls related to processes. Learn to create processes using the **fork** and *exec* family of system calls. Once you've understood how the kernel maintains the metadata of an open file in memory, you'll be able to implement both redirection and pipelines, and handle signals in your programs.

Finally, every user must know the routine tasks related to system administration, and Chapter 19 addresses the basic issues in this domain. Understand the important security features provided by the system. Be familiar with the activities associated with system startup and shutdown, and how file systems are *mounted* and checked for consistency. Also learn to do some elementary backups.

Acknowledgments

I have been ably supported in this endeavor by Kelly Lowery and the team at McGraw-Hill. Many thanks go to the agile Melinda Bilecki who has been actively involved ever since the project was conceived. Apart from organizing and analyzing the reviews, she was almost everywhere, offering suggestions wherever possible. If this book appears both concise and comprehensive, credit must also go to the reviewers who were quick to point out what is relevant and what is not.

Laurie Janssen deserves praise for the way she handled the art work in general, and the cover design in particular. I must also thank Dawn Bercier for the marketing arrangements that she was responsible for. You should see a more informative and useful website with this edition, and full marks for this go to Eric Weber. Finally, I can't but admire Peggy Lucas who managed the production process with confidence and patience. There have been many others who couldn't be mentioned by name, but have contributed just the same.

Final Words of "Wisdom"

Most examples have been tested on Solaris and Linux, but I can't guarantee that they will run error-free on every system. UNIX fragmentation makes sweeping generalizations virtually impossible. If some commands don't work in the way specified in this text, don't conclude that the system has bugs. Nevertheless, bugs in these examples are still possible, and I welcome ones (along with your suggestions at *sumitabha@vsnl.com*) that you may hit upon.

Before I take leave, a note of caution would be in order. Many people missed the UNIX bus through confused and misguided thinking and are now regretting it. Let this not happen to you. Once you have decided to exploit UNIX, you'll learn to build on what's already provided without reinventing the wheel. Sooner rather than later, you'll find a world of opportunity and excitement opening up. Approach the subject with zeal and confidence; I am with you.

Sumitabha Das

Conventions Used in This Book

The key terms used in the book (like **regular expression**) are shown in a bold font. Apart from this, the following conventions have been used in this book:

- Commands, internal commands and user input in examples are shown in bold constant width font:

  Many commands in **more** including **f** and **b** use a repeat factor.
  The shell features three types of loops—**while**, **until**, and **for**.
  `Enter your name:` **henry**

- Apart from command output, filenames, strings, symbols, expressions, options, and keywords are shown in constant width font. For example:

  Most commands are located in `/bin` and `/usr/bin`.
  Try doing that with the name `gordon lightfoot`.
  Use the expression `wilco[cx]k*s*` with the `-l` option.
  The shell looks for the characters `>`, `<` and `<<` in the command line.
  The `-mtime` keyword looks for the modification time of a file.

- Machine and domain names, email addresses, newsgroups, and URLs are displayed in italics:

  When henry logs on to the machine *uranus* .....
  User henry on this host can be addressed as *henry@calcs.planets.com*.
  The newsgroup *comp.lang.perl* discusses problems related to **perl**.
  Executables for all UNIX flavors are available at *http://www.perl.com*.

- Place-holders for filenames, terms, and explanatory comments within examples are displayed in italics:

  Use the `-f` *filename* option if this doesn't work.
  This process has a *controlling terminal*.
  `$` **`cd ../..`** *Moves two levels up*

The following abbreviations, shortcuts and symbols have been used:

- SVR4—System V Release 4
- `sh`—Bourne shell
- `csh`—C shell
- `ksh`—Korn shell
- `$HOME/`*flname*—The file *flname* in the home directory
- `~/`*flname*—The file *flname* in the home directory
- `foo`, `bar`, and `foobar`—Generic file and directory names as used on Usenet
- .... for lines that are not shown
- This box ␣ indicates the space character.
- This pair of arrows ⇤⇥ indicates the tab character.