YOUR UNIX
THE ULTIMATE GUIDE

SUMITABHA DAS

Your UNIX

The Ultimate Guide

Sumitabha Das
Internetworking Consultant

Preface

The limits of my language mean the limits of my world.

—Ludwig Wittgenstein

In an article in *The Bell System Technical Journal* published in 1978, Ken Thompson and Dennis Ritchie introduced to the world a "general-purpose" operating system offering a "high degree of portability." That operating system was UNIX. It has come a long way since then, and it is firmly entrenched today in its famed position as the environment of choice for serious work. Whether it's used for computer-aided design or simulation in the laboratory, UNIX has been the platform that has aided their development. UNIX gave birth to the C language. Enterprise-scale database applications run on UNIX. UNIX systems have also been behind the creation of *Jurassic Park* and *Titanic*. Today, electronic commerce is fueled by UNIX.

However, the ride to fame hasn't always been smooth. UNIX has constantly drawn admirers and detractors alike. Beginners feel overwhelmed by its sheer weight and are often unable to comprehend why UNIX behaves in a radically different manner when compared with the Windows platform. Even experienced computer professionals feel lost in this mysterious universe, characterized by its arcane tools and cryptic syntax. Windows isn't like this at all; why was UNIX designed that way?

There's a reason behind all this. UNIX was never designed for the world. A group of programmers created it to run their own programs—programs that no one else required. But the fact that it still managed to gain wide acceptance without any serious marketing effort points to UNIX's inherent strengths. Because it is rooted in open standards, UNIX has shown up in every hardware and now powers the Internet.

Even though UNIX is based on a few simple ideas, it is extremely rich in features—features never seen or experienced before. UNIX has redefined what an operating system should be and why many people need to master it. The UNIX "command line," with its myriad options and complex syntaxes, drove many people away. But these features and the capability of these commands to act in combination are precisely what make UNIX seem irreplaceable. This is what this book attempts to explain and advise: *There is a method to this madness.*

A single textbook is never adequate to present all these features, but fortunately I could select from this ocean those essential concepts that require specific attention. This book makes no attempt to list all 24 options of the **ls** command or all of the shell's

features. But it does discuss the ones that I consider important and shows you how to apply them to real-world situations. I have treated these concepts in some detail, which will benefit the beginner as well as the expert.

This book is well suited for use in a variety of UNIX courses related to introductory programming, operating systems and system administration—at both the introductory and more advanced levels. A knowledge of one operating system is preferable even though the book makes no such assumption. However, the programming features of the shell can be better understood by someone with a background of one programming language.

Why You Must Know UNIX Today

As I see it, UNIX had to grow through four important phases in its life cycle. Initially, it was considered a product for the engineering and scientific community. The initiated couldn't live without it; the others stayed away because it wasn't "friendly" enough. A lot of what we see in present-day UNIX is the result of the development work of these universities and research institutes.

Next, the corporates took to UNIX. UNIX made significant inroads into large corporations and government organizations. Its robustness established it as the operating system of choice for database work. Every RDBMS package, be it Oracle or Sybase or Informix, has long been available on the major UNIX platforms. If you wanted to be a good DBA (database administrator), you had to know UNIX.

The Internet is the third (and most significant) phase of the UNIX cycle. Many of the things you see on the Net and the Web owe their origin to the hard and inspiring work done by the UNIX community. TCP/IP was first ported to UNIX systems. Most servers on the Net are UNIX machines. Internet Service Providers use UNIX machines. **perl**, which operates behind all the forms that you see on the Web, is a UNIX product. In short, the Internet is UNIX, and to understand its workings and exploit this technology for local internets, one *must* know UNIX.

These years have also seen the silent but phenomenal growth of the Linux cult. Linux is the free UNIX that has captivated the new generation of college students and is now invading the commercial world. It is widely favored for its richness of Internet-related tools. I see a great future ahead for Linux and feel compelled to include its important features in this book. Even though Linux doesn't use any of the original AT&T code, it's just as UNIX as any other flavor—in spirit, in philosophy—in everything.

Why You Should Actually Like UNIX

I am distressed by one trend that seems to emerge from recent adoptions of UNIX. Many people are now entering UNIX through the "back door"—and this back door is the X Window system—the graphical windowing system offered on UNIX machines. These people use UNIX in the same way they handle DOS or Windows and expect the mouse to do all the work. They configure the system by filling up forms of menu-based GUI applications, and they don't even care to know the names of the files or which entries are affected. Let's make no mistake; you can't learn UNIX this way. It takes little time to get disillusioned and join the group of detractors.

Necessity apart, UNIX is actually fascinating. The system is based on its *commands*, programs that are meant to do specific jobs. Commands use *options*, and

each option makes the command behave differently. Many of these commands can be combined with one another (as *filters*) to perform complex tasks. Some of them even use a powerful pattern matching feature (called *regular expressions*) that is used in locating and substituting text. Add to all this a scripting language, and you have a *tool kit* that sets no limits on the things that you can do with this system.

UNIX commands are generally noninteractive, and hence ideally suited for the development of automated systems. You can do things in UNIX at the drop of a hat— things which you simply can't do in the windowing environment. Ask a Windows user to convert a thousand files from BMP to GIF format; he'll run away. If you know your UNIX well, you can do that by writing just three or four lines of simple code. You don't *need to click your mouse a thousand times* (actually more).

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 inno- vate. 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 the Book Is Organized

UNIX versions can be broadly divided into two different schools—the *System V* school from AT&T Bell Laboratories and the *Berkeley* school from the University of Califor- nia, Berkeley. More specifically, versions tend to be looked at as being either based on *SVR4* (System V Release 4—AT&T's last release before winding up its UNIX opera- tions) or on *BSD UNIX* (the UNIX created by Berkeley). This books attempts to por- tray UNIX generically, keeping an eye on the adoption of its features by Solaris. Linux is primarily BSD-based and has been treated in separate sections.

The subject matter is organized differently from what you would find in other textbooks. Though a system component is often spread across multiple chapters, you'll find like material grouped together for ease of understanding and reference. Once you understand this organization, locating a feature or a command shouldn't be difficult:

Topic	Chapters
Editors	4 and 5
File system	6, 7 and 21
Shell	8, 17, 18 and 19
Process	10 and 22
TCP/IP and the Internet	11, 13, 14, 23 and 24
Regular expressions	4, 5, 15, 16 and 20
Filters	9, 15, 16 and 20
System and Network Administration	21, 22, 23 and 24

Allocation of a command or feature to a particular chapter is based on *definite* principles. You probably won't find a separate chapter on file attributes in other text- books, but I have considered it necessary to maintain this arrangement. That's why the basic commands that act on file content (like **cp**, **rm**, **lp**, **compress** and **gzip**) have been separated from those that change or match the attributes only without disturbing the

contents (like **ln**, **chown**, **chgrp** and **find**). The first group can be found in Chapter 6, while Chapter 7 exclusively handles file attributes.

In a similar manner, the shell makes its appearance in three distinct forms: as an interpreter (Chapter 8), as an environment customizer (Chapter 17) and as a programming language (Chapters 18 and 19). I haven't made the mistake of discussing shell programming and job control in a single chapter as some people have done. I feel that job control belongs to the realm of processes, even if it is a feature of the shell. That's why you'll find it in Chapter 10 along with its other companions that display, kill and schedule processes.

Now that you understand the basic overarching structure of the book, it's time you had a look at the organizational details of each chapter:

Chapter 1 takes you on a tour through two hands-on sessions to let you grasp some of the important features of the system. Try out some of the commonly used UNIX commands. Learn to use your keyboard when the system behaves unpredictably. This exposure, when set against the background, also gives you an indication of things to come. Learn the essential features of UNIX from this chapter.

Chapter 2 prepares you for understanding the syntax of UNIX commands—the various forms its options and arguments can take. Learn to use the on-line UNIX documentation, especially the **man** command. The various facilities available in the system not only let you find out what a command does but also locate the command that does a specific job.

Chapter 3 discusses some useful stand-alone utilities. Some of these commands report on the system's parameters like the date (**date**) or the machine name (**uname**). You'll learn to change your password (**passwd**) and set your terminal's characteristics (**stty**). Many of these commands have been used in command pipelines and shell scripts that are featured later in the book.

Chapter 4 presents comprehensive coverage of the **vi** editor, the most popular text editor used on UNIX systems. Get to know the three modes and the commands associated with each mode. Apart from performing the basic editing functions, learn how to use a number with a command to repeat it. Knowledge of searching and substitution techniques also leads you to the first discussion on *regular expressions*. The advanced features include configuration issues and are taken up in the "Going Further" section at the end of the chapter.

Chapter 5 presents yet another editor—GNU **emacs**, which is not universally available but is widely used. Learn the use of the *[Ctrl]* and the *[Meta]* keys in framing command sequences, and the fully-worded commands when no key bindings are available. Use *regions* and windows for productive editing and understand the superior pattern search and substitution techniques. Learn to use the *kill ring* for retrieving multiple sections of copied and deleted text. Customization of the editor and its advanced features are discussed in the end-of-chapter "Going Further" section.

Chapter 6 discusses files and directories. Understand the use of *pathnames* in describing the parent–child relationship between them. Create and remove directories (**mkdir** and **rmdir**) and navigate the file system (**pwd** and **cd**). Learn to create (**cat >**), copy (**cp**), remove (**rm**) and print (**lp**) files. Find out how free your disk is (**df**) and compress your files (**compress**, **gzip** and **zip**) to release space. No file attributes are discussed in this chapter.

All file and directory attributes are discussed in Chapter 7. Use the **ls** command with numerous options to display them. Change the permissions (**chmod**) and ownership (**chown** and **chgrp**) of a file. Understand how permissions acquire a different meaning when applied to directories. Use *links* (**ln**) to provide additional names to a file without changing the *inode number*. Finally, use the **find** command to locate any file or directory in the file system by specifying one or more file attributes. A file's contents are not disturbed in this chapter.

Chapter 8 is the first of four chapters reserved for the shell. Learn the use of *metacharacters* to match multiple filenames with a single pattern. Redirect the input and output *streams* of many commands to originate from or go to another file or another command. Use quotes and the \ to remove the special meaning of these metacharacters. The chapter also introduces the use of shell variables and shell scripts.

Chapter 9 develops on the stream-handling features of the shell to present a family of commands (*filters*) that use streams both for input and output. These are simple filters that manipulate content in a limited way, like extracting sections from different regions of a file (**head, tail, cut** and **paste**). Learn to sort a file (**sort**) and translate a character (**tr**) into something else. Finally, use these commands in combination to have a first taste of the much-hyped tool-building feature of UNIX.

The process management system comes next (Chapter 10). Find out how similar files and processes are, and discover three types of commands in the system when looked at from a process perspective. Understand how processes are created with **fork** and **exec**, display their attributes (**ps**) and kill a runaway process (**kill**). Learn to run jobs in the background (**nohup** and &) and move them between background and foreground (**bg** and **fg**). Also learn to change their priority (**nice**) and even schedule them (**at, batch** and **cron**) for later operation.

We venture into the TCP/IP networking world in Chapter 11. Get introduced to *hostnames*, *IP addresses* and *domain names*. Understand the role played by *ports* and *sockets* to move a packet from one host to another. You'll learn to log on to a remote machine with **telnet** and **rlogin**. Also copy files between machines with **ftp** and **rcp**. Use **rsh** to run a command remotely without logging in.

The GUI in UNIX comes next (Chapter 12) with a minimal discussion on the X Window system. Understand how X is ideally suited for working in a TCP/IP network. X clients need a separate program (the *window manager*) to control the look and feel of its windows. Use the **xterm** client as the launching pad to run all applications. Configure X to behave differently on startup (.xinitrc) and customize its *resources*.

Chapter 13 is all about electronic mail. Understand how the SMTP and POP protocols are used to handle mail. Here, apart from the standard **mail** command, use two menu-based programs, **elm** and **pine**. Forward your mail (.forward) or leave behind messages when you are away (**vacation**). Configure and use Netscape Messenger as a superior mail handling tool. Learn to compose a group of messages offline, use the address book and set up *spam* control. Mail can now handle multimedia attachments as a *multipart* message. Understand *MIME* theory in the "Going Further" section.

Chapter 14 extends Chapter 11 to focus on the Internet and the World Wide Web. Learn how the top-level *domains* are organized. Use a *mailing list*. Discover a domain-like structure in *newsgroups*, and access Net News using **tin** and Netscape. Communicate with multiple persons using the *Internet Relay Chat*. Use *URLs*, *HTTP* and

HTML to access the Web. Customize Netscape Navigator to make Web browsing very productive.

Filters make another appearance in Chapter 15 with two advanced text manipulators. Search for a pattern with commands of the **grep** family (also **egrep** and **fgrep**). Display lines containing and not containing the pattern, and their frequency of occurrence. Use **sed** as a multipurpose filtering tool and especially to substitute one pattern for another. Here, we have the most exhaustive presentation of regular expressions discussed in a phased manner in three sections.

awk as a filter and a programming language makes its appearance in Chapter 16. Break up a line into fields and manipulate each field individually. Use the comparison operators and decimal numbers for computation. Make decisions and iterate within a loop in true programming language style. Use the common **awk** variables and its built-in functions, especially those that relate to string handling. You'll need all this knowledge to understand **perl**, which uses many of **awk**'s features.

The UNIX shell provides excellent opportunities to customize your environment (Chapter 17). Use its variables to set the command search path, the terminal and the prompt. Use the *history* facility to edit and re-run previous commands with simple keystrokes. Devise *aliases* for frequently used commands, and let the shell *complete* filenames and command names for you. Learn to place all customized settings in a startup file. The discussions are presented separately for the Bourne shell, C shell, Korn shell and bash.

Shell programming is introduced in Chapter 18. Develop both interactive (**read**) and noninteractive scripts by passing arguments ($1, $2 etc.). Use the *exit status* of a command to control program flow ($?) and the | | and && operators to act as simple conditionals. Learn to use **test** to check strings and the file attributes. Use the wild-card handling feature of **case** as a superior string checking tool. Exploit the $0 parameter to make a script do different things depending on the name by which it is called. Use the standard programing constructs **if**, **while**, **until** and **for** to develop three interesting scripts.

The shell's advanced features are presented in Chapter 19 and include the special features of the Korn and bash shells. Put values into positional parameters (**set** and **shift**). Have another look at stream redirection with the *here document* and learn to merge streams using special symbols. Use **export** to make variable values available in sub-shells. Learn to use arrays and shell functions. Evaluate a command line twice (**eval**) to develop scripts using numbered prompts and variables. Handle multiple streams using **exec**. Finally, use **trap** to determine script behavior when it receives a signal.

We encounter **perl** in Chapter 20 as the finest and most powerful filter in the UNIX world. Use the *default variable* $_ for condensing many statements. Split a line into a list of variables or an array. Use a nonnumeric subscript with the associative array. **perl** uses all the regular expressions discussed so far, but it uses some of its own too (the final discussion). Learn to handle external files and develop subroutines. **perl**'s dominant role in Internet CGI programming is discussed in the chapter's "Going Further" section using a simple form-based HTML application.

The file system comes up for the last time in Chapter 21—this time from the system administrative point of view. We discuss the device files and their unique attributes. Know all about *partitions, file systems* and *inodes.* Know the features of the var-

ious types of file systems and learn to *mount* and *unmount* them. Check the integrity of a file system with the **fsck** command.

General system administration is taken up in Chapter 22. Learn about the powers and privileges of the super user. Maintain user accounts and groups. Enforce security and learn two special permission modes of a file, the *sticky bit* and *set-user-id*. Understand the role of **init** in the startup and shutdown procedure, and grasp the significance of the fields in /etc/inittab. Make backups with **tar** and **cpio**. The end-of-chapter "Going Further" section discusses printer administration and the scripts used by **init**.

We turn to network administration in Chapter 23 and understand how IP addresses are allocated in a TCP/IP network. Learn to configure your network interface (**ifconfig**), routing (**route**) and display the network statistics (**netstat**). Understand how **inetd** controls the **ftp**, **telnet** and POP services. Connect your machine to the Internet with the *Point-to-Point Protocol* (PPP) using two tools, **dip** and **chat**. Make your directories and file systems shareable using the *Network File System* (NFS).

Finally, use your Linux machine to set up three important Internet services for a fictitious network—name service (DNS), email and Web (Chapter 24). Set up the master, slave and caching-only server using BIND 8. Understand how **sendmail** works, and configure the important parameters in /etc/sendmail.cf to set up standard mail server configurations. Understand the utility of *aliases* for forwarding mail and use **fetchmail** to download mail from a remote POP server. Also set up the **httpd** Web service using Apache. Learn to control CGI script execution, *virtual hosts* and directory access.

How This Book Is Different

At the outset, let me maintain that I made no conscious decision to make this book different from others. Facing a UNIX box was my first encounter with computers—even before I knew what an operating system was. I had no expectations, no sides to take and no one to offer me guidance. Having been out there "in the cold," I feel 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. The implementation of this idea, spurred by the instant delight that I developed in the subject, automatically sets this book apart from others. There are five important points to consider:

1. Clarity of Expression

UNIX concepts are sometimes abstract, and when they are not, their relevance to the real world is not often appreciated. I believe that every concept has to be dissected properly to expose its design considerations. Why was the feature conceived in the first place? Where can one apply it? Is the standard explanation clear? Do the examples leave behind gray areas of understanding? If we don't have positive answers to all these questions, then we are most certainly in a state of ambiguity and confusion.

This book makes sure every concept is explained the way it needs to be. Take, for instance, one of the basic tenets of the UNIX system, *standard output*. We are told what standard output is, but I have not been impressed with the way it has been explained in

the books. On the other hand, I felt that the sequence **who > newfile** is better com-
prehended in this way *(p. 239)*:

> The shell looks at the >, understands that standard output has to be redirected,
> opens the file **newfile**, writes the stream into it and then closes the file. And all
> this happens without **who** knowing absolutely anything about it!

How many people know for a fact that a command has no knowledge of
the source of its input and output? Or that there are important implications involved
when choosing to use **wc < foo** in preference to **wc foo** *(p. 240)*? When do you
need to make a command ignorant of the source of its input *(p. 247)*? Does the \ in
echo "Enter your name \c" really remove a special meaning as it is normally known
to do *(p. 237)*? You must know the answers to these questions before you satisfy your-
self that you have understood the shell.

To take another example, why do I often hear that the command line
find /home -name index.html -print is difficult to remember? Only because many
people don't care to split **find**'s arguments into three components *(p. 218)*:

> **find** *path_list selection_criteria action*
>
> • First, it recursively examines all files in the directories specified in *path_list*.
>   Here, it begins the search from /home.
> • It then matches each file for one or more *selection_criteria*. This always con-
>   sists of an expression in the form *-operator argument* (-name index.html).
>   Here, **find** selects the file if it has the name index.html.
> • Finally, it takes some *action* on those selected files. The action -print simply
>   displays the **find** output on the terminal.

Take a look at Table 7.5 *(p. 220)* and you'll find the selection criteria separated
from the action, and presented in roughly the same sequence the **ls -l** command pres-
ents the file attributes. I haven't seen it organized this way in any other book, and yet I
strongly feel that a power user of **find** needs to look at the command in this way.

There are so many other things that you need to know and yet aren't properly
explained in the standard books. This book explains why file ownership is so important
(p. 208) and how to use the sticky bit to implement group projects *(p. 680)*. You'll dis-
cover three important application areas where links can be used *(p. 215)*. How does the
shell that greets you on login differ from the one that runs your scripts *(p. 523)*? Why
mustn't the **cd** command run in a sub-shell *(p. 298)*? Will a message addressed to
joe winter <winterj@sasol.com> reach him if we misspelled *joe (p. 368—Note)*?
You'll find answers to all these questions (and many, many more) in this book.

2. Both Elementary and Comprehensive

One of my reviewers had earlier observed (before he reviewed this book) that the prob-
lem with writing a good semester-length UNIX book is that you need something ele-

mentary enough for beginning students but comprehensive and "referency" enough for the same students to use as they advance. Making a book that fits into both slots was perhaps the most difficult task I faced, but it seems that this has now been achieved. The final judgment, however, is reserved for you.

GOING FURTHER

This book has an enormous amount of information laid out in a structured manner—much more than can be expected from a book of this size. I have tried to present information that is easy to understand and yet devoid of verbiage. While I didn't deviate from the goal of grouping like material, I also made sure that advanced material was segregated from the essential by locating it at the end of a chapter. Each page there is highlighted with the "Going Further" tab. A beginner should simply ignore these portions during the initial pass.

I felt the need to provide as many examples as I could; in fact, some of the tables contain just examples. Just take a look at the ways **chmod** is used in Table 7.3 *(p. 205)* and discover an undocumented feature! Or the way the shell's wild cards are used with commands in Table 8.1 *(p. 231)*. Since **cron** is an indispensable scheduling tool, the significance of the fields of its configuration file is presented with a good number of examples in Table 10.3 *(p. 313)*. Sometimes, I have combined a concept and an example in the same table. You'll appreciate regular expressions better if you study the entries in Table 15.2 *(p. 441)*.

This book is also a comprehensive reference. Important commands have their options listed in their own tables, and the chapters on **vi** and **emacs** editors have lots of tables. Once you know how to use one editor, you'll find Appendix B *(p. 781)* useful (where **vi** and **emacs** are compared feature for feature) to learn the other. Does the C shell use functions? Or what's the role of the command that is itself named **command** in the Korn and bash shells? Just look up Appendix D *(p. 795)* where all the four shells are compared in detail. We all know that regular expressions are used differently by the major filters, but can you recall whether **grep** uses the \b escape sequence? There's no need to experiment any longer, the regular expression matrix says it all "in one place" *(Appendix C, p. 791)*.

This book is also characterized by the presence of section references everywhere. Each chapter opens with a statement of objectives with pointers to the section numbers. A list of the key terms introduced is also presented at the end with similar references. Both forward and reverse referencing have also been made in the main text:

> If you are afraid that you may accidentally overwrite a file, you can later use your knowledge of aliases *(17.4)* and shell functions *(19.10)* to make **cp** behave in this interactive manner by default.
>
> The same logic has already been implemented by using three commands in a pipeline--**cut**, **sort**, and **uniq** *(9.18.1)*. That one used only a single line of code!

Sometimes, you'll want to know not what a command does, but the command that actually performs a specific function. How does one display lines in double space? Or select those that contain or don't contain a pattern? Does UNIX have any tool that presents lines in reverse order? What are the various copying facilities available? There's a specially prepared HOWTO document *(Appendix E, p. 807)* that you'll find enormously useful for these lookups, as these sample lines will reveal:

How to do or display	Command	Page no.
copy directory tree	cp -r	178
copy file between machines	ftp	329
copy file between machines without authentication	rcp	336
lines containing pattern	grep	434
lines in ASCII collating sequence	sort	275
lines in double space	pr -d -t	267
lines in reverse order	tail -r	272
lines not containing pattern	grep -v	438

If you fail to locate a key term, there's a comprehensive glossary that comes to your rescue *(Appendix G, p. 819)*. Special attention has also been given to the index. In addition to the general index, you have access to specially prepared indexes for the **vi** and **emacs** editors. A comprehensive set of over 200 commands discussed in this book has also been indexed separately. There's also a separate index for the shells that are featured in the book.

3. Realistic Illustrations

These days you'll find UNIX books filled with lots of illustrations, but how many of them truly explain a concept? Even though I probably have not used as many illustrations as some authors have, I have tried to make them more effective. For instance, the following illustration looks at standard output in a rather different way:

F I G U R E 8.2   *The Three Destinations of Standard Output*
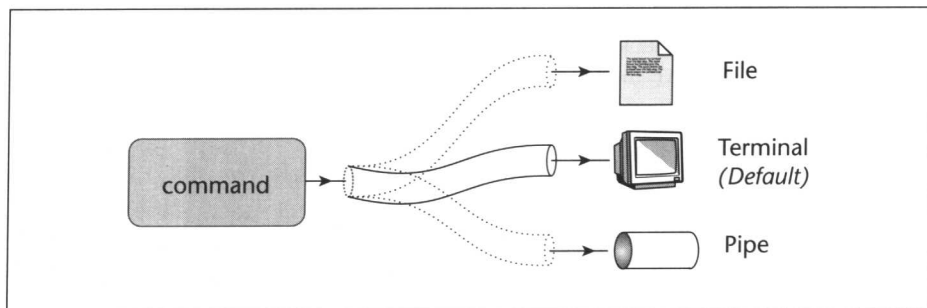
You probably won't have seen standard output portrayed in this manner, but this figure answers many queries. Did it ever occur to you that standard output has three possible destinations? This figure easily replaces a thousand words.

There are many more compelling illustrations. The complexity of the shell's behavioral pattern breaks down when you take a look at Fig 8.1 *(p. 229)*. The process

killing mechanism becomes much easier to comprehend by drawing an analogy with a radar, a gun and an aircraft in Fig 10.2 *(p. 308)*. And you should find the presence of the postman in Fig. 13.7 *(p. 379)* quite meaningful when figuring out how SMTP and POP handle electronic mail.

4. Strong Learning Aids

The idea of reminding a reader about an important topic has always been uppermost in my mind, even though it may have already been mentioned in the text. In fact, these pedagogical aids are a strong feature of this book, and you'll find over 400 instances of such aids in this text. They take on various names, for example, Note, Caution, and Tip. Here's an instance of the first type:

> **Note**
>
> The caret has a triple role to play in regular expressions. When placed at the beginning of a character class (e.g., [^a-z]), it negates every character of the class. When placed outside it, and at the beginning of the expression (e.g., ^2...), the pattern is matched at the beginning of the line. At any other location (e.g., a^b), it matches itself literally.

UNIX is an unusual system; it seldom warns you when you are about to do something disastrous. Yet, the reader would like to be cautioned against taking this disastrous step. If you are going to schedule jobs using the **crontab** command, then this is something you must keep in mind:

> **Caution**
>
> If you use **crontab** - meaning to provide input through the standard input and then decide to abort it, you should terminate it with the interrupt key applicable for your terminal, rather than *[Ctrl-d]*. If you forget to do that, you'll remove all entries from your existing crontab file!

As mentioned before, many of UNIX's mysteries are hidden. A good tip can often save hours of effort, and I have chosen to provide lots of them in this book. Do you really have to quit the **vi** editor every time you want to execute the currently edited shell or **perl** script? No, you don't:

> **Tip**
>
> Can you execute a shell or **perl** script without leaving **vi**? Make this mapping of the function key *[F1]*:
>
> `:map #1 :!%^M`   *File must have executable permission*
>
> This executes (!) the current file (%) at the : prompt. This is one important mapping which the author uses to run scripts without quitting **vi**. You should use it too, and then thank yourself (and the author) when you see how fast a script is debugged. Just remember that **vi** understands the current file as %—a poorly documented feature. If you want to compile the current file from the editor, you'll need to use the % as an argument to the compiling command (**cc** or **javac**, for example).

This book is not about Linux per se, but I consider Linux to be an important member of the UNIX family. In general, Linux commands have more options, and some of them are absolute beauties! I have highlighted features that are either unique to Linux or handled differently by it. Such instances are easily located; just look for the penguin:

Linux

GNU **grep** has a nifty option that locates not only the matching line, but also a certain number of lines *above* and *below* it. For instance, you may want to know what went before and after the **foreach** statement that you used in a **perl** script:

```
$ grep -1 "foreach" count.pl                          One line above and below
print ("Region List\n");
foreach $r_code sort (keys(%regionlist)) {
    print ("$r_code : $region{$r_code} : $regionlist{$r_code}\n") ;
```

The command locates the string foreach and displays one line on either side of it. Isn't this feature useful?

Even though I have used the Bourne shell as the "base" shell in this book, most discussions focus on the other shells—C shell, Korn shell and bash. Rather than have separate chapters for them, I have first discussed a concept in a general manner and then highlighted a shell-specific feature in a separate aside box:

C Shell

The standard error is handled differently by the C shell, so the examples of this section won't work with it. In fact, the C shell merges the standard error with the standard output; it has no separate symbol for handling standard error only.

5. Numerous Questions and Exercises

This book features an enormous number of questions that test the reader's knowledge—over 900 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.

Final Words of "Wisdom"

All examples have been tested with a number of UNIX and Linux systems, but I simply can't guarantee that they will run error-free on every system. UNIX fragmentation makes sweeping generalizations virtually impossible. It is quite possible that some commands may either not be available or may throw out different messages on your

system. You have to take this in your stride, and you need not automatically conclude that the system has bugs. Nevertheless, bugs in these examples are still possible, and I welcome ones (along with all 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. They fell for the mouse, and the mouse couldn't deliver much. Don't let this happen to you. It doesn't have to if you don't want it to—at least, not any longer. Learn to use the tools of the system and build on them without reinventing the wheel. 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* ......
  The RFCs are available at *rs.internic.net*.
  User henry on this host can be addressed as *henry@calcs.planets.com*.
  Every beginner should subscribe to *news.announce.newusers*.
  Download the plugin software from *http://www.shockwave.com*.

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

  Use the -f *filename* option if this doesn't work.
  ..... to develop a set of standard rules (*protocols*)
  We'll ignore the *C* header. *STIME* shows the time the process started.
  Use *Edit>Preferences* to configure Netscape.
  $ **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
- *ksh93*—The ksh93 version of the 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.

Acknowledgments