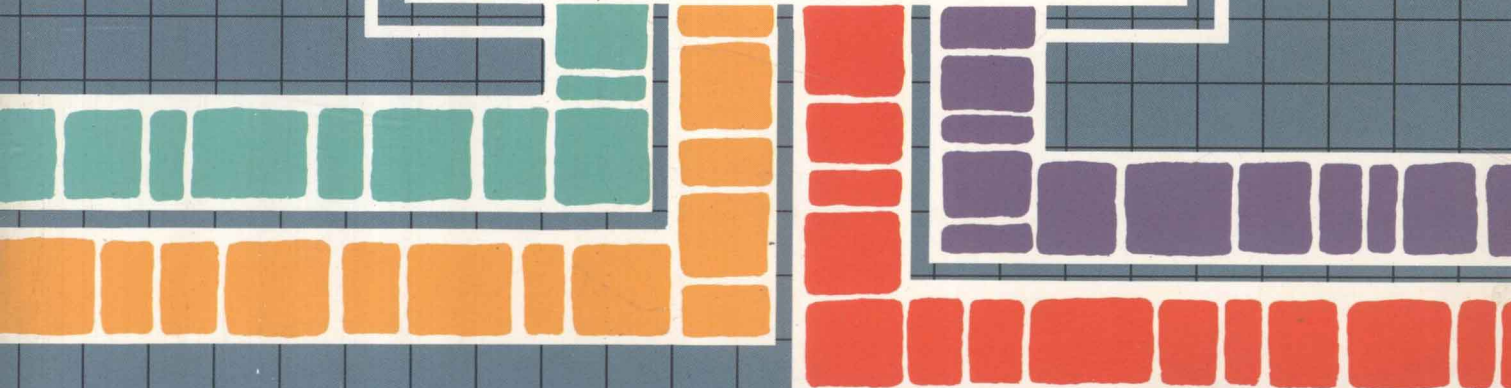


# UNIX<sup>TM</sup> Survival Guide



Elizabeth A. Nichols  
Sidney C. Bailin  
Joseph C. Nichols

# UNIX<sup>TM</sup>

# Survival

# Guide

Elizabeth A. Nichols  
Sidney C. Bailin  
Joseph C. Nichols

**Holt, Rinehart and Winston**

New York • Chicago • San Francisco • Philadelphia  
Montreal • Toronto • London • Sydney  
Tokyo • Mexico City • Rio de Janeiro • Madrid

**Library of Congress Cataloging-in-Publication Data**

Nichols, Elizabeth Agnew.

The UNIX® survival guide.

includes index.

1. UNIX® (Computer operating system) I. Bailin, Sidney C. II. Nichols, Joseph C., 1943-

III. Title.

QA76.76.063N53 1986 005.4'3 85-24758

**ISBN 0-03-000773-9**

Copyright © 1987 by CBS College Publishing

Address correspondence to:

383 Madison Avenue

New York, N.Y. 10017

All rights reserved

Printed in the United States of America

Published simultaneously in Canada

7 8 9 066 9 8 7 6 5 4 3 2 1

CBS College Publishing

Holt, Rinehart and Winston

The Dryden Press

Saunders College Publishing

# PREFACE

It is not controversial (or even very interesting) to state at this time that UNIX is the most popular operating system for super-microcomputers and that its availability is increasing on minicomputers and even mainframes. A fact that we, the authors of this book, feel is very significant is that large classes of UNIX users are becoming increasingly hard to distinguish from UNIX system administrators. In particular, the great majority (in terms of numbers of systems) of UNIX systems are super-microcomputer based. Typically, the group of users of such systems is charged with maintaining the health and well-being of their UNIX system—in addition to using it for their work-related application. In the past, this responsibility would reside in a staff of specialists in a data processing division under the aegis of "UNIX Support" or "UNIX System Administration Staff". Thus, users of "early UNIX" systems (say, pre-1980) had an option of remaining blissfully ignorant of the nitty-gritty details surrounding such activities as:

- Booting the system
- Recovering from such events as power failures and hardware malfunctions
- Backing up system files
- Analyzing and resolving common system problems
- Maintaining user accounts and configuring user working environments

and many more.

It is our contention that such is no longer the case. An ignorant UNIX/super-micro user cannot remain blissful for long. There are additional classes of UNIX users for which this statement is true:

- New UNIX system administrators charged with the health and well-being of any UNIX system (super-micro based or otherwise)
- UNIX users who are responsible for the integrity of their specific application running under UNIX
- UNIX users who wish to augment their knowledge concerning standard tools and techniques that potentially could save much trouble and heart-break.

The *UNIX Survival Guide* is written precisely to address the above audience's information requirements.

Therefore, the book's organization is centered around the problems that such individuals face, from initial delivery of a new system to its on-going (smooth and not-so-smooth) operation. After covering some topics of cultural and/or philosophical interest on the "essence" of UNIX in Chapter 1, we immediately address the concepts and operational procedures for booting a newly delivered UNIX system in Chapter 2. This necessitates covering the bare essentials for understanding the UNIX terminal interface and key shell utilities and commands. Immediately thereafter, we launch into a detailed discussion of booting UNIX from a standard release tape. This illustrates a key difference in our approach from many other texts on UNIX. Our focus is on individuals who do more than just ask user support for a login id and then use the typical complement of shell commands that require no special privileges to invoke. Indeed, our target audience will most likely have access to the super user account and all the power and responsibilities associated therewith.

Certainly, the knowledge needed by UNIX end users and UNIX system administrators has a broad intersection. End users and administrators must both know about such topics as

- The UNIX hierarchical file system
- Text editors under UNIX such as *vi*
- Useful UNIX shell tools
- Shell facilities such as pipes, filters, and input/output redirection
- UNIX facilities for generating processes and for interprocess communication

Where we have attempted to distinguish this text from others is in our depth of coverage of the above topics and the audience to which our treatment is directed. For example, we address the UNIX file system in two chapters—one that concentrates on its hierarchical structure, UNIX file names, and shell commands for positioning within the hierarchy, and one that looks "under the hood" at the physical format used to store a file system on disk. Specifically, we discuss detailed layouts of the superblock, inodes, the free list, and similar bit-level details from the standpoint of its theoretical operation and *most important* from the standpoint of using tools to watch it operate and even repair it when it fails.

Thus, our intent is to cover UNIX on four levels:

- Its *philosophical elegance* as a product of Bell Laboratories research in computer science
- Its basic operation from a user's point of view
- Its tools and facilities for system administration
- Its inner workings—to the extent that this knowledge can be the basis for achieving smoother day-to-day operation

In writing this book over the past two years we have received critical support from SGS Semiconductor Corporation. SGS not only commissioned the writing of this book, but also provided support for the project in the form of a Z8000-based UNIX system on which to produce it, many hours of support, advice, and encouragement on the structure and content of the text, plus endless patience and forbearance as obstacles were met and overcome. In particular, we would like especially to thank Mr. Jean-Claude Monney of SGS/Phoenix and his support staff, who were with us from the beginning.

# CONTENTS

## CHAPTER 1: The UNIX Operating System 1

- 1.1 Hierarchical File System 2
- 1.2 Compatible File, Device, and Interprocess I/O 3
- 1.3 Asynchronous Processes (Multi-tasking) 3
- 1.4 System Command Language Selectable for Each User 4
- 1.5 Portability 4
- 1.6 UNIX Bad Attributes 4

## CHAPTER 2: Getting Started 9

- 2.1 For Systems Administrators Only 10
  - 2.1.1 UNIX Delivered on Tape 10
  - 2.1.2 Booting UNIX from Disk 11
  - 2.1.3 File System Health Check 14
  - 2.1.4 Establishing Terminal Ports 18
  - 2.1.5 Creating New Accounts 22
  - 2.1.6 Setting the Date 23
  - 2.1.7 The rc File 23
  - 2.1.8 Multi-User Mode and Rebooting or Shutting Down the System 24
- 2.2 Scenario for a First UNIX Session
  - 2.2.1 Logging in (Login) and Logging off (CTRL-d) 25
  - 2.2.2 Establishing a Password: passwd 26
  - 2.2.3 Date and Time 27
  - 2.2.4 Who am I and Who Else Is On the System 28
  - 2.2.5 The UNIX Terminal Interface 28
  - 2.2.6 Automatic Setup of Your UNIX Environment: .profile 32
  - 2.2.7 Abort the Current Process (or, the PANIC Button) 32
  - 2.2.8 Flow Control for Terminal Displays: CTRL-q 33
  - 2.2.9 Writing to Other Terminals 35
  - 2.2.10 Echoing Character Strings: echo 38
  - 2.2.11 Introduction to Process Forking: ps and & 38
  - 2.2.12 Hunt the Wumpus 40

## CHAPTER 3: Files and Directories 45

- 3.1 Real-life Files and UNIX Files 46
- 3.2 The UNIX File Hierarchy 46
  - 3.2.1 Directories 47
  - 3.2.2 Absolute (Fully Qualified) File Names 47
  - 3.2.3 The Current Working Directory and Relative File Names 49
- 3.3 File and Directory Naming Conventions 52
- 3.4 Climbing Trees: pwd and cd 54
- 3.5 Looking at Directories: ls 55
- 3.6 Looking at Text Files: cat 61
  - 3.6.1 Copying Files: cp 63
  - 3.5.2 Renaming Files: mv 66
- 3.7 Generating Alias File Names: ln 68
  - 3.7.1 Deleting Files: rm 68
- 3.8 Directory Manipulation: rmdir and mkdir 70
- 3.9 Changing File Access Permissions: chmod 71
  - 3.9.1 Numeric Mode Specification 71
  - 3.9.2 Symbolic Mode Specification 72
- 3.10 Changing File Ownership: chown 74
- 3.11 File Name Generation: The Special Characters ?, \*, [, ], and - 74
  - 3.11.1 Single Character Wildcard: ? 75
  - 3.11.2 String Wildcard: \* 76
  - 3.11.3 Sets of Characters: [] and - 76
  - 3.11.4 File Name Generation and Invisible Files 77
  - 3.11.5 The Shell Command Line Interpreter: A Peek Under the Hood 78



## CHAPTER 4: The vi and ed Editors 81

- 4.1 Line and Screen Editors 81
- 4.2 Setup for vi 82
  - 4.2.1 Terminal Characteristics: TERM, CAP 82
  - 4.2.2 Execution Characteristics: EXINIT 83
    - 4.2.2.1 Checking EXINIT Options 85
    - 4.2.2.2 Setting EXINIT Options 86
- 4.3 vi: Getting In and Out 87
  - 4.3.1 How vi Manages Text 87
  - 4.3.2 Invoking vi (Getting In) 89
  - 4.3.3 vi Command Language: General Topics 92
  - 4.3.4 Terminating a vi Session 93
  - 4.3.5 Insert and Replace Commands 94
- 4.4 Cursor Positioning: Line, Character, Window, Text-Object 98
- 4.5 Operators: Line and Character 102
- 4.6 File Manipulation Commands 108
  - 4.6.1 The Remembered File Name 109
  - 4.6.2 Saving Text from Memory to Disk: w 110
  - 4.6.3 Reading Text from Disk to Memory: r, e, and n 110
- 4.7 Cutting and Pasting: Yank and Put 111
  - 4.7.1 Scope: Sentences, Paragraphs, and Buffers 111
  - 4.7.2 Moving Text Blocks 116
  - 4.7.3 Copying Text Blocks 116
  - 4.7.4 vi Buffers: Additional Facts 117
- 4.8 Recovery: Undoing What You Have Done 118
- 4.9 Recovery from System Crashes 119
- 4.10 The Line Editors *ed* and *ex* 120
- 4.11 vi Command Summary 122

## CHAPTER 5: I/O, The UNIX File System, and I/O Redirection 129

- 5.1 More on the UNIX File System 129
  - 5.1.1 Special Files 130
  - 5.1.2 Special File Directory Entries 133
  - 5.1.3 Raw and Cooked Modes of I/O 134
  - 5.1.4 Pipeline Files 135
  - 5.1.5 Major Device Types and Minor Device Numbers 135
- 5.2 File Descriptors as Logical Device Addresses 136
- 5.3 Shell I/O Redirection Commands 137
- 5.4 Examples 138

## CHAPTER 6: Processes and Signals 143

- 6.1 Concepts 143
  - 6.1.1 Processes and Program Text 143
  - 6.1.2 Concurrency of Processes: Time-Sharing 147
  - 6.1.3 Signals 147
  - 6.1.4 Forking: How Processes Are Created 153
  - 6.1.5 The Hierarchy of Processes, The Scheduler, and Init 154
  - 6.1.6 Cron and Crontab 155
  - 6.1.7 Restarting a Hung Terminal 156
  - 6.1.8 Real and Effective User ids 159
- 6.2 Commands: ps and kill 161
  - 6.2.1 ps 161
  - 6.2.2 kill 163

## CHAPTER 7: Pipes and Filters 165

- 7.1 Concepts 165
- 7.2 Examples 170

## CHAPTER 8: Utilities and Recipes Grab Bag 175

- 8.1 File Print Commands 176
  - 8.1.1 Displaying a File Page by Page: more 176
  - 8.1.2 Displaying a File with Headers, Columns, Page Numbers: pr 178
  - 8.1.3 Displaying Pieces of File: tail 180
- 8.2 File Manipulation Commands 181
  - 8.2.1 Determining File Type: file 181
  - 8.2.2 File Comparison: cmp, comm, and diff 182
  - 8.2.3 Finding/Printing Files, Directories and Subdirectories: find 188
  - 8.2.4 Sorting Lines in Files: sort 190
  - 8.2.5 Checking Validity of File Content: sum 193
- 8.3 Pattern Search Utilities 194
  - 8.3.1 Regular Expressions (REs) 194
  - 8.3.2 The grep Family: grep, fgrep, and egrep 204
  - 8.3.3 awk – Pattern Scanning and Processing Language 208

## 8.4 The Terminal Capabilities Facility 216

8.4.1 Setting Up to Use TERMCAP 218    8.4.2 Terminal Capabilities 219    8.4.3 TERMCAP Definition  
Implementation Notes 226    8.4.4 The TERMCAP Terminal Driver Routines 228

## CHAPTER 9: The UNIX File System 231

### 9.1 UNIX File Systems: Introduction 231

9.1.1 Inodes 231    9.1.2 The Free List 231    9.1.3 The Superblock 233    9.1.4 The Boot Block 234    9.1.5  
Directories 234    9.1.6 Mounting of File Systems 236    9.1.7 Reconstructing the Free List 240    9.1.8  
Pseudodisks 241    9.1.9 Creating a File System (*mkfs*) 242    9.1.10 Non-Standard File Systems 243

### 9.2 UNIX File Systems: Details 244

9.2.1 Detailed Structure of the Superblock 244    9.2.2 Detailed Structure of an Inode 247

### 9.3 Backing It All Up ... Or Part of It 248

9.3.1 *Dd* 248    9.3.2 *Tar* 251    9.3.3 *Dump/Restor* 253    9.3.4 *Cpio* (Systems III and V) 258    9.3.5  
*Volcopy* (Systems 3 and 5) 259

### 9.4 Useful Utilities 261

9.4.1 Counting Used and Free Blocks (*df*, *du*, *quot*) 261    9.4.2 Inspecting and Modifying Files: (*od*, *adb*, *clri*,  
*izap*) 263

### 9.5 How to Fix a File System in Version 7

(or, an icheck a day keeps the dup blocks away) 269

9.5.1 Link Count Consistency 269    9.5.2 Missing, Duplicate, and Nonsensical Block Numbers 271

### 9.6 For Detectives Only: A Tour Through Some System Tables 273

## CHAPTER 10: Generating New UNIX System 279

### 10.1 Configuring and Generating Version 7 281

10.1.1 The System Parameter File: *Param.h* 281    10.1.2 The Kernel Source Modules (Excluding Device  
Drivers): */usr/sys/\** 283    10.1.3 The Device Driver Source Modules: */usr/sys/dev* 283    10.1.4 The  
Configuration Files: *conf*, *c.c*, and *l.s* 284    10.1.5 Compiling the UNIX Kernel 285

### 10.2 Configuring and Generating Systems 3 and 5 286

## CHAPTER 11: UNIX Overview - Revisited 289

### 11.1 The Building Block Approach 289

### 11.2 Uniformity *Uber Alles* 289

11.2.1 External Uniformities 290    11.2.2 Internal Uniformities 293    11.2.3 Programming Uniformity 294

### 11.3 Bye (or CTRL-d) 294

## APPENDIX A 295



# The Unix Operating System

# 1

"Intelligence ... is the faculty of making artificial objects, especially tools to make tools."—Bergson

The above quote appears in the Foreword to *The Bell System Technical Journal*, July-August 1978. This issue of the BSTJ was dedicated to articles about the "hot," "new" UNIX Operating system. All the UNIX luminaries contributed articles:

- Ritchie and Thompson wrote a general overview.
- Bourne wrote on the UNIX shell.
- Ritchie, Kernighan, Lesk, and Johnson wrote on the C Language.
- Lesk, Osanna, and Kernighan wrote on Document Preparation.
- .... etc.

What is UNIX? Why is there such a frenzy in the microcomputer and minicomputer industries to jump onto the UNIX bandwagon? In this chapter we attempt to provide you with some answers, without assuming that you know anything about UNIX—other than that it is a significant development in the computer industry.

The quote at the beginning of the chapter captures some of the essence of UNIX. It is an operating system that makes it easy to build tools and software applications. UNIX makes heavy use of *abstract concepts* to identify what is common among diverse software constructs. This commonality is expressed in small, specialized, and necessary functional modules that creative users can combine to form more complex functions. What UNIX provides so effectively is, first, the "right" choice of modules; and, second, the "glue" necessary to make the pieces work together.

On a less philosophical level, UNIX is a timesharing system that provides:

- A hierarchical file system.
- Compatible file, device, and inter-process I/O.
- The ability to initiate asynchronous processes.
- A system command language selectable on a per user basis.
- A high degree of portability.

The above bullets are the classical UNIX attributes. They were originally presented by D.M. Ritchie and K. Thompson in a talk to the Fourth ACM Symposium on Operating System Principals, in October 1973. In the following paragraphs we briefly describe each of these attributes, and we comment on their relative importance as contributors to the wide popularity of UNIX.

### 1.1. Hierarchical File System

A hierarchical file system provides users with a simple way of grouping related files together. This can be extremely useful when there are hundreds of files to manage. For example, with a hierarchical file system you can identify general categories such as letters, projects, months, or clients. You can then subdivide each general category into meaningful subcategories. The general category "projects" would be broken down into the names of individual projects. Under each particular project would be the documents relating to that project. And so on.

File hierarchies are especially desirable for managing software development. A typical UNIX software development project might have its files organized as in Figure 1-1.

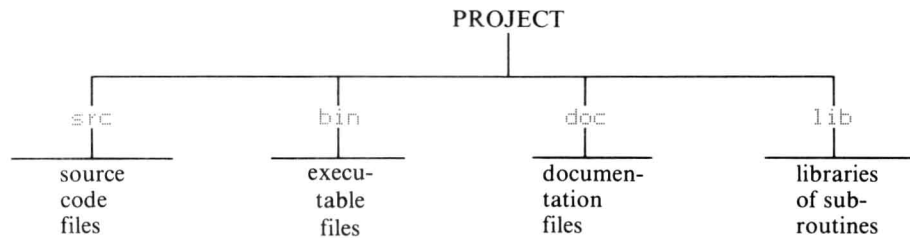


Figure 1-1: File Hierarchy of a UNIX Software Development Project

Suppose for example that we have a project to develop some database software for a customer. Let us call this project `DB_SW`. Suppose one of the programs being developed is called *new\_item*—it is the program that lets you add an item to the database. And suppose (just for kicks) that this program is written in the C language.

We might store the C-language source code as `/DB_SW/src/new_item.c`. The binary executable (machine-language) form of the program would probably be stored as `/DB_SW/bin/new_item`. In UNIX terminology these are examples of *pathnames*.

The pathname `/DB_SW/src/new_item.c` refers to a file called *new\_item.c*, which is listed in a directory called *src*, which itself is listed in a directory called `/DB_SW`. The pathname `/DB_SW/bin/new_item` refers to a file called *new\_item*, which is listed in a directory called *bin*, which is listed in the directory `/DB_SW`.

To support this structure, UNIX provides a command we can use to list the files in any particular directory: for example, to list the files in the directory `/DB_SW`, or the files in the directory `/DB_SW/bin`, or those in `/DB_SW/src`.

In contrast to this consider the CP/M\* operating system. Under CP/M, a user invoking the *dir* command sees a list of *all* the files on a disk, with no grouping into logical categories. This is all right if there are not many files: in CP/M, you would typically see one, two, or maybe three screenfuls of file names.

But in a large project, or in an office where there are many projects going on, the number of files can easily exceed this. Most UNIX systems have large enough disks to hold many more files than a typical CP/M system would. Listing them all, you might see as many as thirty screenfuls of file names. The capability to group these files into a logical hierarchy then becomes immensely valuable.

\*CP/M is a trademark of Digital Research, Incorporated.

## 1.2. Compatible File, Device, and Interprocess I/O

This concept is difficult to appreciate without having actually used UNIX and seen its power and effectiveness. "I/O" stands for "input and output." Compatibility of file, device, and interprocess I/O is an example of UNIX's *abstraction* of common computer functions.

Consider a small function, such as the function that sorts lines in a file. The sort program accepts input from some source, processes it somehow, and then outputs the (possibly altered) results. The source for the data could be a file, or a device, or another program; and the same is true for the destination of the sorted data. Figure 1-2 illustrates these possibilities.

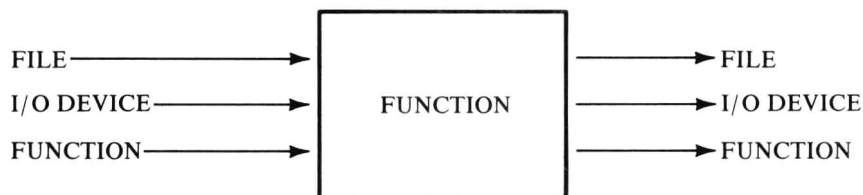


Figure 1-2: Possible Sources and Destinations of Data

To take a specific example, suppose the sort program reads a set of names and addresses from an "input file" and prints out a list sorted by last name. The output *could* be printed; but alternatively it could be written to another "output file." Another interesting possibility is when the output serves as input to another program—this is called *interprocess I/O*. This second program might format the sorted lines so they can be printed on pages of 66 lines each. (The formatting might involve inserting headings and page numbers.)

UNIX does not distinguish between these three possible data sources and three possible destinations. A user can specify at execution time which ones apply to the current situation. Moreover, the command syntax for naming files, devices, and processes (programs) is the same.

As you become more familiar with UNIX, this consistency among files, devices, and processes will likely become one of your most heavily used (and coveted) UNIX features.

## 1.3. Asynchronous Processes (Multi-tasking)

A *multi-tasking* system allows you to run more than one program at a time from a single terminal. These simultaneously running programs are called *tasks* or *processes*. Multi-tasking is not synonymous with *multi-user*. A *multi-user* system is one that supports more than one terminal user concurrently. UNIX is both a multi-user and a multi-tasking operating system.

The idea of multi-tasking is that certain programs do not need to "talk" to the user while they run. These programs can operate invisibly in the *background*; while a single *foreground* task is free to write data to the terminal, accept commands from the terminal, and so on.

Multi-tasking is certainly very useful for software developers. A typical application is to begin compiling several programs in the background and then edit another program in the foreground.

But what about end users? Can they benefit from multi-tasking? We think the answer is a definite *yes*. A typical problem with single-tasking systems is the delay incurred when printing a document. A user must wait for the document to be printed before he can execute the next program. In a multi-tasking system, the print program can be placed in the background while the user proceeds with other functions in the foreground.

#### 1.4. System Command Language Selectable for Each User

In UNIX, the command language by which the user "talks" to the computer is called a *shell*. There are two popular standard shells that are delivered with UNIX systems: the Bourne shell and the C shell. The examples in this book all use the Bourne shell. One quick way to tell if your shell is of the Bourne or C variety is to enter the following command:

```
> filename
```

If no error is returned, then you are probably using the Bourne shell. However, this is not a fool-proof test because there are a number of shells being distributed that combine features from both dialects. These hybrid shells are not easily classified.

The essential point here is that a system administrator or developer can install alternative command languages under UNIX. In particular, users requiring a specialized custom language can have that language as their shell. And users requiring a simple "non-computereze" language (perhaps a set of menus from which options can be selected) can have *that* language as their shell. Associated with each user's account is the name of the program that acts as his shell. As we mentioned, the standard, default shell for UNIX from AT&T is the Bourne shell, but this can be changed easily.

#### 1.5. Portability

Portability is perhaps the most important contributing factor to the widespread success of UNIX. As hardware technology continues to progress at an ever faster pace, the problem of software obsolescence—the high cost of rewriting software to run on new hardware—has forced the industry to place high value on portability.

UNIX is written almost entirely in the C language. C is a high-level, machine-independent language that is well suited for "system" software—the kind of software that goes into an operating system. UNIX is the prime example of this, but recently other operating systems have been written in C.

Moving UNIX to a new computer is greatly simplified because most of it is machine-independent. Only a small amount of the UNIX code needs to be modified when it is transported to a new machine. Moreover, as the C language becomes more and more popular, UNIX becomes more and more portable.

As a real-life example of portability in the commercial market place, the authors have ported a relational data base management system (called UNIFY) to MC68000\*-based microcomputers, to the Pyramid Reduced Instruction Set Computers (RISC), and to the Gould/SEL Concept 32 series of superminis, using approximately one man-week of effort each. The database package consists of over 1600 modules, totalling over 150,000 lines of source code. The number of man-years that went into the original creation of this package probably exceeds 10.

To summarize, UNIX is so "hot" today because it permits users to benefit from the continuing advances in computer hardware without giving up the software that they have used in the past.

#### 1.6. UNIX Bad Attributes

After such a glowing description, what is there to say about UNIX that is less positive? Here are some of the common complaints about UNIX:

---

\*MC68000 is a trademark of Motorola Corporation.

### *UNIX Is Not User-Friendly*

The names of functions in UNIX are not as natural as some would like. For example, the name of a command to display the content of a file on the terminal screen is *cat*. Other names commonly singled out for criticism are *grep*, *awk*, *yacc*, *lex*, *nroff*, *rm*, and many more.

Hardware vendors have reacted particularly strongly to this complaint. The usual solution is to offer a shell based on *menus*, so that the user need not memorize so many unnatural mnemonics.

The problem with menus is that after you have used them several times, they may become tedious, cumbersome, and unnecessary.

### *UNIX Lets You Hang Yourself*

UNIX provides the user with some very, *very* powerful commands which it will silently execute. When invoked by the novice user, these commands can have unforeseen, sometimes disastrous results. For example,

```
rm *
```

removes *all* the files in the user's current directory. Besides the objection to the mnemonic *rm*, UNIX critics believe that before executing such a potentially catastrophic command, UNIX should query the user if he really means it.

In defense of UNIX, there is an option to the remove command (*i*, for *interactive*) that gives the user a yes/no option before deleting each file. Still, even an experienced user might intend to type

```
rm *.old
```

and accidentally hit the carriage-return key right after the *\**. And some valuable files may be lost.

Another example is the copy command, *cp*. If a user issues this command to copy a source file (*fileA*) to a destination (*fileB*), UNIX will silently overwrite *fileB*.

To these complaints, many system developers respond that with power comes responsibility. They prefer to retain the power and flexibility and are unwilling to give these up for protection they do not feel they need. End users will probably not agree.

### *UNIX Is Not Well Suited for Data Base Applications*

UNIX views the disk as a collection of thousands of *blocks*. The blocks contained in any one file are not necessarily contiguous on the disk. Over the course of its life a file may grow or shrink, and it may acquire new blocks or lose some. Thus, the blocks in a file can end up being scattered all over the disk.

Accessing the file then becomes very inefficient. To reach the widely separated blocks, the *disk-head* must constantly move back and forth. (The disk-head is the component that transfers data to and from specific locations on the disk. It is analogous to the tone arm on a record player.) The mechanical movement of the disk-head is much, much slower than the electronic processing within the computer itself. Consequently, accessing files on a disk can slow down the entire system—noticeably.

Obviously, the larger the file, the worse the potential fragmentation problem. And data bases tend to be large files. UNIX can develop severe performance problems in applications requiring a large data base.

There is a rebuttal to this criticism—at least a partial rebuttal. It is sometimes possible to go outside of the normal UNIX file system to store a data base; this depends on the data base management system being used. A facility in UNIX called *raw device I/O* allows you to identify

the data base with a contiguous set of blocks on the disk. Raw I/O provides almost unlimited flexibility in the way the data base is stored on disk.

The problem is that several commercially available data base packages do not use raw I/O. A user is therefore vulnerable to performance problems if the data base becomes larger than, say, one megabyte (one million characters).

#### *UNIX Does Not Handle Error Conditions Well*

Generic UNIX as offered by AT&T does not recover well from bad blocks on a disk. Some enhanced versions of UNIX do better by maintaining a record of bad blocks which are then avoided.

It is best to ensure that your disks are error free when you buy them. Most disk vendors offer guaranteed error-free disks for an extra charge. The cost is well worth it.

#### *UNIX Documentation Stinks*

The standard documentation from AT&T is of uneven quality. Reading it can be a taxing experience even for UNIX gurus. To someone trying to learn UNIX, it is often incomprehensible. What with all the attention now, and especially in light of AT&T's push to commercialize UNIX, we are confident that this problem will soon be resolved.

#### *UNIX Does Not Provide a Very Secure Environment*

For example: a user can write on *any* other user's screen, unless some explicit action is taken to prevent this. A user can prevent others from writing on his screen, but this requires an explicit preventative action—the ordinary state of affairs is permissive.

UNIX was developed under the assumption that all users were operating together in a friendly, team-spirited environment where tradeoffs between convenience (or flexibility) and security were consistently made in favor of the former.

Still, there is a modicum of security in the UNIX environment. Files in UNIX can be given permission attributes that prevent unauthorized access. Each user account can be given a password. You can also limit the range of commands that are available without special authorization. You would do this by substituting a special, limited shell for the Bourne or C shell.

#### *UNIX Is Not a Real-Time Operating System*

Generic UNIX from AT&T does not give you much control over how the computer is shared among competing, concurrently running programs (processes). In particular, UNIX does not provide a way for you to prioritize processes explicitly.

This capability is important for *real-time* applications, which must function under very stringent timing constraints. A number of UNIX systems on the market have addressed this need, but the official package from AT&T does not.

Another useful feature for real-time applications is the sharing of memory between two concurrent programs. This can be a very efficient way for concurrent processes to pass information back and forth. This feature is not supported in Version 7 and System 3, but it *is* in System 5.

It is important to remember that UNIX was never billed as a real-time operating system. It was meant for *time-sharing*: interactive, command-response communication with users at several terminals. The very fact that people now want to use UNIX for real-time applications attests to the universal appeal of the system, despite its limitations in certain areas.

Overall, the market place has determined that the benefits of UNIX outweigh its

drawbacks. In our opinion, the adoption of UNIX as an *ad hoc* standard operating system by the computer industry is a very positive development that will benefit everyone: end users and system developers alike.

In the next chapters of this book we try to provide not only highlights, but also some serious technical detail where we think it helpful. The final chapter then returns to the themes we have just discussed, in an attempt to unify what you have learned and capture the UNIX philosophy in a more specific way.





# Getting Started

# 2

Getting started with UNIX has to be addressed from two viewpoints. First, the viewpoint of the *system administrator*: someone who has perhaps just unpacked and assembled the hardware, has a UNIX boot tape, and is tasked with bringing the system up from scratch. The first section of this chapter (Section 2.1) is dedicated to system administrators. Assuming no prior knowledge about UNIX, we discuss:

- Boot tapes.
- Single user mode.
- Setting the date.
- The *root* and user file systems.
- Multi-user mode.
- Bringing up terminals.
- New user accounts.
- Shutting down the system.
- Checking system health.

The treatment in Section 2.1 is somewhat cookbookish: we do not explain underlying reasons or theory, but try only to *get you started*—to get you a working UNIX system. With this accomplished, you will be able to "walk through" the examples and case studies that we provide throughout the book.

The second viewpoint is that of a *user*, who wants the system for some productive purpose. The user is not concerned with maintaining or managing the system—he just wants to *use* it. He is not concerned with how his account was created, or how to power up the system in the morning, or how to shut it down at night, or how to back up a disk. Those are the system administrator's responsibility. The *users* who are reading this chapter should begin with the second section (2.2).

Thus, while the *administrator* puts in a great deal of effort to arrive at the first *login*: prompt on each terminal, the *user* essentially *begins* his relationship with UNIX at that point.

It is our contention, though, that in many new UNIX shops—especially those using microcomputers—the distinction between administrators and users will become blurred. This can lead to problems. Controlling the system configuration becomes difficult when users start to take on administrative functions. The informality that is common in these microcomputer installations seems to promote this phenomenon, however, so it is best to acknowledge it and attempt to deal with it.

Throughout this book we try to present information that is relevant both to users *and* to system administrators. In this chapter, Section 2.1 is mainly for administrators—or for users who will be performing administrative functions. If you are a prospective user with the luxury of an already running UNIX system, then our advice is get an account and dive into Section 2.2.