CHAPTER 1
FROM TYPEWRITERS TO WORD PROCESSORS

Before we consider the special tools that the UNIX environment provides for text processing, we need to think about the underlying changes in the process of writing that are inevitable when you begin to use a computer.

The most important features of a computer program for writers are the ability to remember what is typed and the ability to allow incremental changes—no more retyping from scratch each time a draft is revised. For a writer first encountering word processing software, no other features even begin to compare. The crudest command structure, the most elementary formatting capabilities, will be forgiven because of the immense labor savings that take place.

Writing is basically an iterative process. It is a rare writer who dashes out a finished piece; most of us work in circles, returning again and again to the same piece of prose, adding or deleting words, phrases, and sentences, changing the order of thoughts, and elaborating a single sentence into pages of text.

A writer working on paper periodically needs to clear the deck—to type a clean copy, free of elaboration. As the writer reads the new copy, the process of revision continues, a word here, a sentence there, until the new draft is as obscured by changes as the first. As Joyce Carol Oates is said to have remarked: “No book is ever finished. It is abandoned.”

Word processing first took hold in the office as a tool to help secretaries prepare perfect letters, memos, and reports. As dedicated word processors were replaced with low-cost personal computers, writers were quick to see the value of this new tool. In a civilization obsessed with the written word, it is no accident that WordStar, a word processing program, was one of the first best sellers of the personal computer revolution.

As you learn to write with a word processor, your working style changes. Because it is so easy to make revisions, it is much more forgivable to think with your fingers when you write, rather than to carefully outline your thoughts beforehand and polish each sentence as you create it.

If you do work from an outline, you can enter it first, then write your first draft by filling in the outline, section by section. If you are writing a structured document such as a technical manual, your outline points become the headings in your document; if you are writing a free-flowing work, they can be subsumed gradually in the text as you flesh them out. In either case, it is easy to write in small segments that can be moved as you reorganize your ideas.

Watching a writer at work on a word processor is very different from watching a writer at work on a typewriter. A typewriter tends to enforce a linear flow—you must write a passage and then go back later to revise it. On a word processor, revisions are constant—you type a sentence, then go back to change the sentence above. Perhaps you write a few words, change your mind, and back up to take a different tack; or you decide the paragraph you just wrote would make more sense if you put it ahead of the one you wrote before, and move it on the spot.

This is not to say that a written work is created on a word processor in a single smooth flow; in fact, the writer using a word processor tends to create many more drafts than a compatriot who still uses a pen or typewriter. Instead of three or four drafts, the writer may produce ten or twenty. There is still a certain editorial distance that comes only when you read a printed copy. This is especially true when that printed copy is nicely formatted and letter perfect.

This brings us to the second major benefit of word-processing programs: they help the writer with simple formatting of a document. For example, a word processor may automatically insert carriage returns at the end of each line and adjust the space between words so that all the lines are the same length. Even more importantly, the text is automatically readjusted when you make changes. There are probably commands for centering, underlining, and boldfacing text.

The rough formatting of a document can cover a multitude of sins. As you read through your scrawled markup of a preliminary typewritten draft, it is easy to lose track of the overall flow of the document. Not so when you have a clean copy—the flaws of organization and content stand out vividly against the crisp new sheets of paper.

However, the added capability to print a clean draft after each revision also puts an added burden on the writer. Where once you had only to worry about content, you may now find yourself fussing with consistency of margins, headings, boldface, italics, and all the other formerly superfluous impedimenta that have now become integral to your task.

As the writer gets increasingly involved in the formatting of a document, it becomes essential that the tools help revise the document's appearance as easily as its content. Given these changes imposed by the evolution from typewriters to word processors, let's take a look at what a word-processing system needs to offer to the writer.

□ A Workspace □

One of the most important capabilities of a word processor is that it provides a space in which you can create documents. In one sense, the video display screen on your terminal, which echoes the characters you type, is analogous to a sheet of paper. But the workspace of a word processor is not so unambiguous as a sheet of paper wound into a typewriter, that may be added neatly to the stack of completed work when finished, or torn out and crumpled as a false start. From the computer's point of view, your workspace is a block of memory, called a *buffer*, that is allocated when you begin a word-processing session. This buffer is a temporary holding area for storing your work and is emptied at the end of each session.

To save your work, you have to write the contents of the buffer to a file. A file is a permanent storage area on a disk (a hard disk or a floppy disk). After you have saved your work in a file, you can retrieve it for use in another session.

When you begin a session editing a document that exists on file, a copy of the file is made and its contents are read into the buffer. You actually work on the copy, making changes to *it*, not the original. The file is not changed until you save your changes during or at the end of your work session. You can also discard changes made to the buffered copy, keeping the original file intact, or save multiple versions of a document in separate files.

Particularly when working with larger documents, the management of disk files can become a major effort. If, like most writers, you save multiple drafts, it is easy to lose track of which version of a file is the latest.

An ideal text-processing environment for serious writers should provide tools for saving and managing multiple drafts on disk, not just on paper. It should allow the writer to

- work on documents of any length;
- save multiple versions of a file;
- save part of the buffer into a file for later use;
- switch easily between multiple files;
- insert the contents of an existing file into the buffer;
- summarize the differences between two versions of a document.

Most word-processing programs for personal computers seem to work best for short documents such as the letters and memos that offices churn out by the millions each day. Although it is possible to create longer documents, many features that would help organize a large document such as a book or manual are missing from these programs.

However, long before word processors became popular, programmers were using another class of programs called *text editors*. Text editors were designed chiefly for entering computer programs, not text. Furthermore, they were designed for use by computer professionals, not computer novices. As a result, a text editor can be more difficult to learn, lacking many on-screen formatting features available with most word processors.

Nonetheless, the text editors used in program development environments can provide much better facilities for managing large writing projects than their office word processing counterparts. Large programs, like large documents, are often contained in many separate files; furthermore, it is essential to track the differences between versions of a program.

UNIX is a pre-eminent program development environment and, as such, it is also a superb document development environment. Although its text editing tools at first may appear limited in contrast to sophisticated office word processors, they are in fact considerably more powerful.

▫ Tools for Editing ▫

For many, the ability to retrieve a document from a file and make multiple revisions painlessly makes it impossible to write at a typewriter again. However, before you can get the benefits of word processing, there is a lot to learn.

Editing operations are performed by issuing commands. Each word-processing system has its own unique set of commands. At a minimum, there are commands to

- move to a particular position in the document;
- insert new text;
- change or replace text;
- delete text;
- copy or move text.

To make changes to a document, you must be able to move to that place in the text where you want to make your edits. Most documents are too large to be displayed in their entirety on a single terminal screen, which generally displays 24 lines of text. Usually only a portion of a document is displayed. This partial view of your document is sometimes referred to as a *window*.* If you are entering new text and reach the bottom line in the window, the text on the screen automatically scrolls (rolls up) to reveal an additional line at the bottom. A cursor (an underline or block) marks your current position in the window.

There are basically two kinds of movement:

- scrolling new text into the window
- positioning the cursor within the window

When you begin a session, the first line of text is the first line in the window, and the cursor is positioned on the first character. Scrolling commands change which lines are displayed in the window by moving forward or backward through the document. Cursor-positioning commands allow you to move up and down to individual lines, and along lines to particular characters.

After you position the cursor, you must issue a command to make the desired edit. The command you choose indicates how much text will be affected: a character, a word, a line, or a sentence.

Because the same keyboard is used to enter both text and commands, there must be some way to distinguish between the two. Some word-processing programs assume that you are entering text unless you specify otherwise; newly entered text either replaces existing text or pushes it over to make room for the new text. Commands are entered by pressing special keys on the keyboard, or by combining a standard key with a special key, such as the *control key* (*CTRL*).

Other programs assume that you are issuing commands; you must enter a command before you can type any text at all. There are advantages and disadvantages to each approach. Starting out in text mode is

*Some editors, such as `emacs`, can split the terminal screen into multiple windows. In addition, many high-powered UNIX workstations with large bit-mapped screens have their own windowing software that allows multiple programs to be run simultaneously in separate windows. For purposes of this book, we assume you are using the `vi` editor and an alphanumeric terminal with only a single window.

more intuitive to those coming from a typewriter, but may be slower for experienced writers, because all commands must be entered by special key combinations that are often hard to reach and slow down typing. (We'll return to this topic when we discuss `vi`, a UNIX text editor).

Far more significant than the style of command entry is the range and speed of commands. For example, though it is heaven for someone used to a typewriter to be able to delete a word and type in a replacement, it is even better to be able to issue a command that will replace every occurrence of that word in an entire document. And, after you start making such global changes, it is essential to have some way to undo them if you make a mistake.

A word processor that substitutes ease of learning for ease of use by having fewer commands will ultimately fail the serious writer, because the investment of time spent learning complex commands can easily be repaid when they simplify complex tasks.

And when you do issue a complex command, it is important that it works as quickly as possible, so that you aren't left waiting while the computer grinds away. The extra seconds add up when you spend hours or days at the keyboard, and, once having been given a taste of freedom from drudgery, writers want as much freedom as they can get.

Text editors were developed before word processors (in the rapid evolution of computers). Many of them were originally designed for printing terminals, rather than for the CRT-based terminals used by word processors. These programs tend to have commands that work with text on a line-by-line basis. These commands are often more obscure than the equivalent office word-processing commands.

However, though the commands used by text editors are sometimes more difficult to learn, they are usually very effective. (The commands designed for use with slow paper terminals were often extraordinarily powerful, to make up for the limited capabilities of the input and output device).

There are two basic kinds of text editors, *line editors* and *screen editors*, and both are available in UNIX. The difference is simple: line editors display one line at a time, and screen editors can display approximately 24 lines or a full screen.

The line editors in UNIX include `ed`, `sed`, and `ex`. Although these line editors are obsolete for general-purpose use by writers, there are applications at which they excel, as we will see in Chapters 7 and 12.

The most common screen editor in UNIX is `vi`. Learning `vi` or some other suitable editor is the first step in mastering the UNIX text-processing environment. Most of your time will be spent using the editor.

UNIX screen editors such as `vi` and `emacs` (another editor available on many UNIX systems) lack ease-of-learning features common in many word processors—there are no menus and only primitive on-line help screens, and the commands are often complex and nonintuitive—but they are powerful and fast. What's more, UNIX line editors such as `ex` and `sed` give additional capabilities not found in word processors—the ability to write a script of editing commands that can be applied to multiple files. Such editing scripts open new ranges of capability to the writer.

▫ Document Formatting ▫

Text editing is wonderful, but the object of the writing process is to produce a printed document for others to read. And a printed document is more than words on paper; it is an arrangement of text on a page. For instance, the elements of a business letter are arranged in a consistent format, which helps the person reading the letter identify those elements. Reports and more complex documents, such as technical manuals or books, require even greater attention to formatting. The format of a document conveys how information is organized, assisting in the presentation of ideas to a reader.

Most word-processing programs have built-in formatting capabilities. Formatting commands are intermixed with editing commands, so that you can shape your document on the screen. Such formatting commands are simple extensions of those available to someone working with a typewriter. For example, an automatic centering command saves the trouble of manually counting characters to center a title or other text. There may also be such features as automatic pagination and printing of headers or footers.

Text editors, by contrast, usually have few formatting capabilities. Because they were designed for entering programs, their formatting capabilities tend to be oriented toward the formats required by one or more programming languages.

Even programmers write reports, however. Especially at AT&T (where UNIX was developed), there was a great emphasis on document preparation tools to help the programmers and scientists of Bell Labs produce research reports, manuals, and other documents associated with their development work.

Word processing, with its emphasis on easy-to-use programs with simple on-screen formatting, was in its infancy. Computerized phototypesetting, on the other hand, was already a developed art. Until quite recently, it was not possible to represent on a video screen the variable type styles and sizes used in typeset documents. As a result, phototypesetting has long used a markup system that indicates formatting instructions with special codes. These formatting instructions to the computerized typesetter are often direct descendants of the instructions that were formerly given to a human typesetter—center the next line, indent five spaces, boldface this heading.

The text formatter most commonly used with the UNIX system is called `nroff`. To use it, you must intersperse formatting instructions (usually one- or two-letter codes preceded by a period) within your text, then pass the file through the formatter. The `nroff` program interprets the formatting codes and reformats the document "on the fly" while passing it on to the printer. The `nroff` formatter prepares documents for printing on line printers, dot-matrix printers, and letter-quality printers. Another program called `troff` uses an extended version of the same markup language used by `nroff`, but prepares documents for printing on laser printers and typesetters. We'll talk more about printing in a moment.

Although formatting with a markup language may seem to be a far inferior system to the "what you see is what you get" (*wysiwyg*) approach of most office word processing programs, it actually has many advantages.

First, unless you are using a very sophisticated computer, with very sophisticated software (what has come to be called an electronic publishing system, rather than a mere word processor), it is not possible to display everything on the screen just as it will appear on the printed page. For example, the screen may not be able to represent boldfacing or underlining except with special formatting codes. WordStar, one of the grandfathers of word-processing programs for personal computers, represents underlining by surrounding the word or words to be underlined with the special control character `^S` (the character generated by holding down the *control* key while typing the letter *S*). For example, the following title line would be underlined when the document is printed:

```
^SWord Processing with WordStar^S
```

Is this really superior to the following `nroff` construct?

```
.ul
Text Processing with vi and nroff
```

It is perhaps unfair to pick on WordStar, an older word-processing program, but very few word-processing programs can complete the illusion that what you see on the screen is what you will get on paper. There is usually some mix of control codes with on-screen formatting. More to the point, though, is the fact that most word processors are oriented toward the production of short documents. When you get beyond a letter, memo, or report, you start to understand that there is more to formatting than meets the eye.

Although "what you see is what you get" is fine for laying out a single page, it is much harder to enforce consistency across a large document. The design of a large document is often determined before writing is begun, just as a set of plans for a house are drawn up before anyone starts construction. The design is a plan for organizing a document, arranging various parts so that the same types of material are handled in the same way.

The parts of a document might be chapters, sections, or subsections. For instance, a technical manual is often organized into chapters and appendices. Within each chapter, there might be numbered sections that are further divided into three or four levels of subsections.

Document design seeks to accomplish across the entire document what is accomplished by the table of contents of a book. It presents the structure of a document and helps the reader locate information.

Each of the parts must be clearly identified. The design specifies how they will look, trying to achieve consistency throughout the document. The strategy might specify that major section headings will be all uppercase, underlined, with three blank lines above and two below, and secondary headings will be in uppercase and lowercase, underlined, with two blank lines above and one below.

If you have ever tried to format a large document using a word processor, you have probably found it difficult to enforce consistency in such formatting details as these. By contrast, a markup language—especially one like `nroff` that allows you to define repeated command sequences, or *macros*—makes it easy: the style of a heading is defined once, and a code used to reference it. For example, a top-level heading might be specified by the code `.H1`, and a secondary heading by `.H2`. Even more significantly, if you later decide to change the design, you simply change the definition of the relevant design elements. If you have used a word processor to format the document as it was written, it is usually a painful task to go back and change the format.

Some word-processing programs, such as Microsoft WORD, include features for defining global document formats, but these features are not as widespread as they are in markup systems.

□ Printing □

The formatting capabilities of a word-processing system are limited by what can be output on a printer. For example, some printers cannot backspace and therefore cannot underline. For this discussion, we are considering four different classes of printers: dot matrix, letter quality, phototypesetter, and laser.

A *dot-matrix* printer composes characters as a series of dots. It is usually suitable for preparing interoffice memos and obtaining fast printouts of large files.

```
This paragraph was printed with a dot-matrix printer.  It uses a print
head containing 9 pins, which are adjusted to produce the shape of each
character.  More sophisticated dot-matrix printers have print heads
containing up to 24 pins.  The greater the number of pins, the finer
the dots that are printed, and the more possible it is to fool the eye
into thinking it sees a solid character.  Dot matrix printers are also
capable of printing out graphic displays.
```

A *letter-quality* printer is more expensive and slower. Its printing mechanism operates like a typewriter and achieves a similar result.

This paragraph was printed with a letter-quality printer. It is essentially a computer-controlled typewriter and, like a typewriter, uses a print ball or wheel containing fully formed characters.

A letter-quality printer produces clearer, easier-to-read copy than a dot-matrix printer. Letter-quality printers are generally used in offices for formal correspondence as well as for the final drafts of proposals and reports.

Until very recently, documents that needed a higher quality of printing than that available with letter-quality printers were sent out for typesetting. Even if draft copy was word-processed, the material was often re-entered by the typesetter, although many typesetting companies can read the files created by popular word-processing programs and use them as a starting point for typesetting.

> This paragraph, like the rest of this book, was phototypeset. In phototypesetting, a photographic technique is used to print characters on film or photographic paper. There is a wide choice of type styles, and the characters are much more finely formed that those produced by a letter-quality printer. Characters are produced by an arrangement of tiny dots, much like a dot-matrix printer—but there are over 1000 dots per inch.

There are several major advantages to typesetting. The high resolution allows for the design of aesthetically pleasing type. The shape of the characters is much finer. In addition, where dot-matrix and letter-quality type is usually constant width (narrow letters like *i* take up the same amount of space as wide ones like *m*), typesetters use variable-width type, in which narrow letters take up less space than wide ones. In addition, it's possible to mix styles (for example, bold and italic) and sizes of type on the same page.

Most typesetting equipment uses a markup language rather than a *wysiwyg* approach to specify point sizes, type styles, leading, and so on. Until recently, the technology didn't even exist to represent on a screen the variable-width typefaces that appear in published books and magazines.

AT&T, a company with its own extensive internal publishing operation, developed its own typesetting markup language and typesetting program—a sister to `nroff` called `troff` (*typesetter-roff*). Although `troff` extends the capabilities of `nroff` in significant ways, it is almost totally compatible with it.

Until recently, unless you had access to a typesetter, you didn't have much use for `troff`. The development of low-cost laser printers that can produce near typeset quality output at a fraction of the cost has changed all that.

> **This paragraph was produced on a laser printer. Laser printers produce high-resolution characters—300 to 500 dots per inch—though they are not quite as finely formed as phototypeset characters. Laser printers are not only cheaper to purchase than phototypesetters, they also print on plain paper, just like Xerox machines, and are therefore much cheaper to operate. However, as is always the case with computers, you need the proper software to take advantage of improved hardware capabilities.**

Word-processing software (particularly that developed for the Apple Macintosh, which has a high-resolution graphics screen capable of representing variable type fonts) is beginning to tap the capabilities of laser printers. However, most of the microcomputer-based packages still have many limitations. Nonetheless, a markup language such as that provided by `troff` still provides the easiest and lowest-cost access to the world of electronic publishing for many types of documents.

The point made previously, that markup languages are preferable to *wysiwyg* systems for large documents, is especially true when you begin to use variable size fonts, leading, and other advanced formatting features. It is easy to lose track of the overall format of your document and difficult to make overall changes after your formatted text is in place. Only the most expensive electronic publishing systems (most of them based on advanced UNIX workstations) give you both the capability to see what you will get on the screen and the ability to define and easily change overall document formats.

□ Other UNIX Text-Processing Tools □

Document editing and formatting are the most important parts of text processing, but they are not the whole story. For instance, in writing many types of documents, such as technical manuals, the writer rarely starts from scratch. Something is already written, whether it be a first draft written by someone else, a product specification, or an outdated version of a manual. It would be useful to get a copy of that material to work with. If that material was produced with a word processor or has been entered on another system, UNIX's communications facilities can transfer the file from the remote system to your own.

Then you can use a number of custom-made programs to search through and extract useful information. Word-processing programs often store text in files with different internal formats. UNIX provides a number of useful analysis and translation tools that can help decipher files with nonstandard formats. Other tools allow you to "cut and paste" portions of a document into the one you are writing.

As the document is being written, there are programs to check spelling, style, and diction. The reports produced by those programs can help you see if there is any detectable pattern in syntax or structure that might make a document more difficult for the user than it needs to be.

Although many documents are written once and published or filed, there is also a large class of documents (manuals in particular) that are revised again and again. Documents such as these require special tools for managing revisions. UNIX program development tools such as SCCS (Source Code Control System) and `diff` can be used by writers to compare past versions with the current draft and print out reports of the differences, or generate printed copies with change bars in the margin marking the differences.

In addition to all of the individual tools it provides, UNIX is a particularly fertile environment for writers who aren't afraid of computers, because it is easy to write command files, or *shell scripts*, that combine individual programs into more complex tools to meet your specific needs. For example, automatic index generation is a complex task that is not handled by any of the standard UNIX text-processing tools. We will show you ways to perform this and other tasks by applying the tools available in the UNIX environment and a little ingenuity.

We have two different objectives in this book. The first objective is that you learn to use many of the tools available on most UNIX systems. The second objective is that you develop an understanding of how these different tools can work together in a document preparation system. We're not just presenting a UNIX user's manual, but suggesting applications for which the various programs can be used.

To take full advantage of the UNIX text-processing environment, you must do more than just learn a few programs. For the writer, the job includes establishing standards and conventions about how documents will be stored, in what format they should appear in print, and what kinds of programs are needed to help this process take place efficiently with the use of a computer. Another way of looking at it is that you have to make certain choices prior to beginning a project. We want to encourage you to make your own choices, set your own standards, and realize the many possibilities that are open to a diligent and creative person.

In the past, many of the steps in creating a finished book were out of the hands of the writer. Proofreaders and copy editors went over the text for spelling and grammatical errors. It was generally the printer who did the typesetting (a service usually paid by the publisher). At the print shop, a typesetter (a person) retyped the text and specified the font sizes and styles. A graphic artist, performing layout and pasteup, made many of the decisions about the appearance of the printed page.

Although producing a high-quality book can still involve many people, UNIX provides the tools that allow a writer to control the process from start to finish. An analogy is the difference between an assembly worker on a production line who views only one step in the process and a craftsman who guides the product from beginning to end. The craftsman has his own system of putting together a product, whereas the assembly worker has the system imposed upon him.

After you are acquainted with the basic tools available in UNIX and have spent some time using them, you can design additional tools to perform work that you think is necessary and helpful. To create these tools, you will write shell scripts that use the resources of UNIX in special ways. We think there is a certain satisfaction that comes with accomplishing such tasks by computer. It seems to us to reward careful thought.

What programming means to us is that when we confront a problem that normally submits only to tedium or brute force, we think of a way to get the computer to solve the problem. Doing this often means looking at the problem in a more general way and solving it in a way that can be applied again and again.

One of the most important books on UNIX is *The UNIX Programming Environment* by Brian W. Kernighan and Rob Pike. They write that what makes UNIX effective "is an approach to programming, a philosophy of using the computer." At the heart of this philosophy "is the idea that the power of a system comes more from the relationships among programs than from the programs themselves."

When we talk about building a document preparation system, it is this philosophy that we are trying to apply. As a consequence, this is a system that has great flexibility and gives the builders a feeling of breaking new ground. The UNIX text-processing environment is a system that can be tailored to the specific tasks you want to accomplish. In many instances, it can let you do just what a word processor does. In many more instances, it lets you use more of the computer to do things that a word processor either can't do or can't do very well.