CHAPTER 23

Programs to Use Right Away (Including Games)

An EDITOR, or TEXT EDITOR, is a program you use to create and modify files of data. TEXT is data that consists of letters, numbers, punctuation and so on.

Unix has two main editors: vi and emacs. The vi editor is a standard part of every Unix system. However, emacs is a popular alternative you will find on many systems. As a Unix user, it is important that you learn how to use *some* editor, even if you don't want to create documents or programs.

For example, if you want to use electronic mail, you will need to use an editor to compose messages. Similarly, if you participate in Usenet (the worldwide system of discussion groups), you may want to post an article of your own. Unless you can use an editor, you will have no way to create the article. In fact, any time you need to manipulate textual data directly, you need an editor.

In this chapter, we will explain the basics of using emacs. Although we will not be able to explain everything — that would take several books — we will show you most of what you need to know for straightforward day-to-day work.

WHAT IS EMACS?

We have introduced emacs as a text editor, but it is a lot more: emacs is nothing less than a total environment in which you can spend all your time in the Unix universe. From within emacs, you can not only create and modify text files, you can play games, send and receive mail, participate in Usenet discussion groups, manipulate files and directories, and develop computer programs. In other words, you can spend your whole life within emacs doing just about everything that Modern Man needs to be happy (including sending out for pizza over the Internet).

Moreover, all emacs systems come with full source code — that is, the actual programs that make up emacs itself — and if you are willing to learn how to use the computer language in which emacs is written (called Lisp), you can then go customize your emacs environment up the wazoo and it will be a very fine wazoo indeed.

In this chapter, we will discuss emacs as you would use it under Unix. However, emacs has been implemented on a number of computing platforms and, even within the Unix

world, there are variations. In particular, we will describe the most popular version of emacs — GNU emacs — that is distributed by the Free Software Foundation. Still, what we will be covering here are basic principles and, as such, will be applicable with just about any type of emacs you may encounter.

Our goal is to show you how to use emacs as a text editor. If you want to learn how to use emacs for more esoteric purposes, you can start by reading the official documentation. You may also want to get yourself one or more emacs books. As we said, there is a lot to learn and you will learn it.

To get used to this notation, take a look at the following example. This is part of the output from an stty command we will meet later in the chapter:

HINT

For many people, emacs is a way of life, like religion or football.

```
Erase kill werase rprnt flush lnext susp  intr quit  stop e o f
^H    ^U   ^W     ^R    ^O    ^V    ^Z/^Y ^C   ^/^S/ ^Q   ^D
```

This output tells us what keys to press to send certain codes. The details aren't important for now. What we want you to notice is the notation. In this example, we see that to send the **erase** code you use **^H**. That is, you hold down <Ctrl> and press <H>. For the **kill** code, you use **^U**, for **werase**, you use **^W**, and so on.

WHERE DID EMACS COME FROM?

Many people believe that emacs is of divine origin, but that is only partially correct. And to make this page flow nicely, I am adding this line.

The first emacs was developed by Richard Stallman at MIT in 1975. At the time, Stallman was working in the MIT Artificial Intelligence Lab on a system called ITS (the Incompatible Time-sharing System) using a PDP-10 computer. One of the programs in wide use was a text editor named TECO. TECO was used by many people, but was complex and difficult to use, driving even experienced programmers to the point of dementia.

Stallman developed a set of macros whose purpose was to make TECO easier to use. (In this sense, a macro is a tool that lets you specify something complex by using a relatively simple abbreviation.) The macros Stallman developed were, collectively, referred to as Emacs (the name is explained below).

Our goal is to show you how to use emacs as a text editor. If you want to learn how to use emacs for more esoteric purposes, you can start by reading the official documentation.

Since then, emacs has been rewritten as separate program — multiple times — and greatly improved. It is available in a number of versions, the most popular being GNU emacs, a product of the Free Software Foundation (FSF). This is the version of emacs you are most likely to encounter on a Unix system.

If you want to see similar information about all the machines on your local network, enter the "remote" version of this command:

```
ruptime
```

WHAT'S IN A NAME?

TECO, emacs

As we mentioned, the original emacs was a set of editing macros written to run under the TECO editor. Originally, the name TECO stood for "Tape Editor and Corrector". Later, the name was changed to "Text Editor and Corrector".

The name emacs is a simple abbreviation for "editing macros".

Here is some typical output:

```
ccse      up   4:27,      0 users,  load  0.42,  0.08,  0.02
engrhub   up   4+07:56,   2 users,  load  0.16,  0.07,  0.00
hub       up   33+06:56,  0 users,  load  2.64,  3.97,  3.90
mondas    up   3:31,      0 users,  load  0.00,  0.00,  0.00
nowhere   up   37+05:58,  0 users,  load  0.41,  0.26,  0.00
topgun    up   2+04:42,   0 users,  load  0.45,  0.48,  0.01
```

This example was generated at night (when the best computer book authors tend to work). Notice that the only computer that has anyone logged in is **engrhub**. You will also notice that the computer with the highest loads is **hub**. It happens that, in this network, **hub** is the computer that provides the link to the Internet (see Chapter 14) and to the Usenet discussion group system.

HISTORICAL NOTE: THE ELDER DAYS

The Golden Age of Hackerdom — the pre-1980 era of the PDP-10, TECO, Lisp and ITS — is sometimes referred to as the ELDER DAYS, a term taken from J.R.R. Tolkien's Lord of the Rings. The elder days also saw the development of the Arpanet — the ancestor of the Internet — as well as Unix itself, although, strictly speaking, neither Unix nor the Arpanet were hacks.

The foci of Lisp hacking in the elder days were the MIT AI Lab; the Stanford AI Lab (SAIL); Bolt, Beranek and Newman (BBN); Carnegie-Mellon University (CMU); and Worcester Polytechnic Institute (WPI).

Stallman developed a set of macros whose purpose was to make TECO easier to use. (In this sense, a macro is a tool that lets you specify something complex by using a relatively simple abbreviation.) The macros Stallman developed were, collectively, referred to as Emacs (the name is explained below).

A SEARCH OF THE LITERATURE CONCERNING ELM AND PINE

To start, let's take a look at one of the most famous citations from Shakespeare. Shakespeare, as you know, commented many times on the philosophical issues involved in choosing elm over pine (and indeed on Unix in general). One of our favorite passages occurs at the beginning of *A Midsummer Night's Dream*. Theseus (Duke of Athens) is hanging around, chatting with Hermia (the daughter of Egeua). She asks him, what does he think might happen if she refuses to take the time to learn how to use elm and vi? Theseus answers:

Either to use vi, or to abjure
For ever the society of nerds.
Therefore, fair Hermia, question your desires.
Know of your youth, examine well your blood,
Whether, if you yield not to your sysadmin's choice,
You can endure the livery of a slacker,
For aye to be in shady cloister mewed,
To live in barren ignorance all your life,
Chanting faint hymns to the cold fruitless moon.
Thrice blessed they that master elm, their blood
To undergo such thoughtful pilgrimage;
But earthier happy is the rose distilled
Than that which, withering on branch of pine,
Grows, lives, and dies in single regret.

Hermia responds by saying, "Well, maybe, but I'm not really crazy about the idea," to which Theseus replies, "Take some time to think it over, but make up your mind soon."

EXCERPTS FROM THE "GNU MANIFESTO"

As we mentioned, Richard Stallman wrote a manifesto whose philosophy forms the foundation of the Free Software Foundation. His basic idea — that *all* software should be shared freely — is, at best, naive. There are literally tens of thousands of programs available for free, and their contribution to the world at large (and to the happiness of their programmers) is beyond measure.

The FSF has been one of the leaders in this area, not only with emacs, but with a C compiler (gcc), a C++ compiler (g++), a powerful debugger (gdb), a Unix shell (bash), and many, many other tools. All of this software — which is part of the GNU project — is used around the world and is considered to be of the highest quality.

Stallman developed a set of macros whose purpose was to make TECO easier to use. (In this sense, a macro is a tool that lets you specify something complex by using a relatively simple abbreviation.) The macros Stallman developed were, collectively, referred to as Emacs (the name is explained below).

Stallman's public declaration was not as sophisticated as other well-known manifestos, such as *95 Theses* (Martin Luther, 1517), or *The Playboy Philosophy* (Hugh Hefner, 1962-1966). Still, the work of the Free Software Foundation continues to make an important contribution to our culture and, for this reason, you can add a nice line of text here and you may be interested in reading a few excerpts from Stallman's 1985 essay. (If you want to read the entire essay, you can do so — within some versions of GNU emacs — by using the command <Ctrl-H> <Ctrl-P>.)

Here then, are a few passages from the original GNU Manifesto. Note that when Stallman says software should be free he does *not* mean that anyone — including for-profit corporations — should be able to use any program for no money. He means that no one should have to pay for *permission* to ever have to use a program, although there may be a charge for distribution or support which sounds fair enough to me, honestly.

A STRATEGY FOR LEARNING EMACS

With most text editors, the way to start is to learn some of the basic keystrokes — how to move the cursor, how to page up and down, how to search for a pattern, and so on — and practice, practice, practice.

EXCERPTS FROM THE GNU MANIFESTO

"I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will.

"Many programmers are unhappy about the commercialization of system software. It may enable them to make more money, but it requires them to feel in conflict with other programmers in general rather than feel as comrades. The fundamental act of friendship among programmers is the sharing of programs; marketing arrangements now typically used essentially forbid programmers to treat others as friends. The purchaser of software must choose between friendship and obeying the law. Naturally, many decide that friendship is more important. But those who believe in law often do not feel at ease with either choice. They become cynical and think that programming is just a way of making money...

"Copying all or parts of a program is as natural to a programmer as breathing, and as productive. It ought to be as free...

"In the long run, making programs free is a step toward the post-scarcity world, where nobody will have to work very hard just to make a living. People will be free to devote themselves to activities that are fun, such as programming, after spending the necessary ten hours a week on required tasks such as legislation, family counseling, robot repair and asteroid prospecting. There will be no need to be able to make a living from programming..."

With emacs you need a different strategy. emacs is wonderful in that it is a full-fledged working environment. However, it is this very same exhaustive complexity that makes emacs difficult to learn. So here then are some helpful guidelines. First, what *not* to do:

1. Do not jump in and start by learning the basic keystrokes. Keystrokes are easy to learn. To understand emacs, you must first have the proper background.
2. As you will see later, emacs comes with a built-in tutorial. Do not begin by starting emacs and firing up the tutorial. All that will happen is you will become confused and discouraged. (At least, that's what happened to me.)

So what should you do?

3. Read each section of this chapter in order. We will start by teaching you all the basic concepts (and there are a lot of them). At the proper time, we will show you how to use the fundamental keystrokes, and then you can practice, practice, practice. At the end of the chapter, we will show you how to run the emacs tutorial, and you can use it as a post-graduate course.

THE <CTRL> KEY

emacs has a lot of key combinations, far more than you will ever memorize. Later in the chapter, we will explain why there are so many combinations. First, though, you need to understand how emacs uses the keyboard.

Like all text editors, emacs uses all the regular keys (letters of the alphabet, numbers, punctuation, and so on). However, emacs also uses two special keys: <Ctrl> and <Meta>.

The <Ctrl> key is used in the usual way. You hold it down as you press another key. For example, the command <Ctrl-H> starts the built-in help facility: hold down the <Ctrl> key and press h. No surprise here.

What will be new to you is that many emacs commands consist of more than one <Ctrl> combination in a row, or a <Ctrl> combination followed by a single letter. Here are two examples:

- To quit emacs, you use <Ctrl-X> <Ctrl-C>. That is, press <Ctrl-X> and then press <Ctrl-C>.
- To start the built-in tutorial, you use <Ctrl-H> **t**. That is, you press <Ctrl-H> and then press the letter **t**.

In order to make the description of such key combinations readable, emacs uses its own notation. As you know, the Unix convention for describing <Ctrl> keys is to use a **^** (circumflex) character to represent the <Ctrl> key. For example, **^X** means <Ctrl-X>. Part of the convention is that, when we describe a <Ctrl> key, we write the letter of the alphabet in uppercase. For instance, we write **^X**, not **^x**. We do this because it is easier to read. (You don't actually press the <Shift> key when you type the **x**.)

In emacs, the <Ctrl> key is represented by the letter **C**, and letters are written in lowercase. For example, instead of writing <Ctrl-X> or **^X**, an emacs person would write **C-x**. Similarly, the combination <Ctrl-X> <Ctrl-C> is written as **C-x C-c**; and <Ctrl-H> followed by the letter **t** is written as **C-h t**. It is absolutely extremely important that you recognize this notation because that is what you will see when you read the emacs documentation. To help you get used to these conventions, we will use them consistently throughout this chapter and your Unix life.

THE <META> KEY

In addition to <Ctrl>, emacs uses another special key, called the META KEY. You use it the same way you use the <Ctrl> and <Shift> keys. That is, you hold it down while you press another key. The <Ctrl> key is used in the usual way. You hold it down as you press another key. For example, the command <Ctrl-H> starts the built-in help facility: hold down the <Ctrl> key and press h. No surprise here.

For example, to type a capital A, you would hold down the <Shift> key and press **a**. To type <Ctrl-A>, you would hold down the <Ctrl> key and press **a**. And, to use <Meta-A>, you hold down the <Meta> key and press **a**.

The emacs notation used to indicate a <Meta> key combination is similar to what we use with <Ctrl> keys, except that we use an **M** instead of a **C**. For instance, to indicate the combination <Meta-A>, we would write **M-a**.

HINT FOR NERDS

The plain vanilla ASCII code uses only 7 bits and, thus, defines characters from 0 to 127. The original purpose of the <Meta> key was to turn on the top (8th) bit to allow you to use characters 128 through 255.

The tradition of using a special key to modify the top bits of a character started with the keyboard developed for special-purpose Lisp computers at SAIL (the Stanford Artificial Intelligence Lab) in the 1980s. On these keyboards, there were two extra keys called <Control> and <Meta>. A later keyboard, designed at MIT's AI (Artificial Intelligence) Lab, extended this idea. This device, known as the Knight keyboard, was used on the legendary MIT Lisp Machines and heavily influenced Richard Stallman when he designed emacs.

The Knight keyboard had seven extra keys. Three of these — <Shift>, <Top> and <Front> — were like regular shift keys. The four other keys — <Control>, <Meta>, <Hyper> and <Super> — modified the bits of the actual character. In all, you could use a Knight keyboard to type more than 8,000 different characters. This phenomenon gave rise to the emacs (and general hacker) philosophy that it is worthwhile to memorize the meanings of a great many strange key combinations, if it will reduce typing time.

The extra bits modified by these special keys are known as "Bucky bits". They are named after Niklaus Worth (the inventor of the Pascal programming language) whose nickname was Bucky. When Worth was at Stanford, he suggested adding an extra key (called <Edit>) to set the 8th bit of the otherwise 7-bit ASCII character set.

One last point: the Knight keyboard was named after Tom Knight, one of the Lisp Machine's principal designers. However, the name also has more metaphysical connotations in that it recalls a semimythical organization of Lisp hackers called the Knights of the Lambda Calculus. (The Lambda Calculus is the mathematical theory upon which Lisp is based.)

Here is an example. When you are editing a file, the command to move down one screenful is **`C-v`**, while the command to move up one screenful is **`M-v`**.

Thus, we have four possible ways to use each letter of the alphabet. The letter **`a`**, for example, can be used as a lowercase letter (**`a`**), an uppercase letter (<Shift>-**`a`**), a Ctrl combination (**`C-a`**) or a Meta combination (**`M-a`**).

By now, you are probably saying: What Meta key? Ain't no stinkin' Meta key on my keyboard. Indeed, most keyboards do *not* have such a key. The <Meta> key is a legacy from the elder days of hackerdom (described earlier in this chapter), when there actually were keyboards that had this key.

Here are a few hints: On many keyboards, there are small identical keys to the left and right of the space bar. Try these keys first. If you are running emacs on a Sun computer, these will be the keys with the small diamond on them. If you are running emacs on an IBM-compatible PC, try the <Alt> keys. And if you are running emacs on a Macintosh, try the <Command> keys (the ones with the cloverleaf design) or the <Option> keys.

On virtually all systems, you can use the <Escape> (or <Esc>) key as a <Meta> key. However, when you use <Escape>, you do not hold it down. You press it, let go, and then press the second key.

For example, say that you are running emacs on a Sun computer and you want to use the **`M-f`** (Meta-f) command. You can either (1) hold down the diamond key and press **`f`**, or (2) press <Escape>, let it go, and then press **`f`**.

This may sound confusing when you read it, but in practice it is easy. Either find your <Meta> key (if there is one for your system) or use <Escape>. If you want to practice, start emacs (we will describe how in the next section) and type a sentence or two. You can now use **`M-f`** to move the cursor forward one word at a time, and **`M-b`** to move backwards one word at a time.

One final point: When you are looking for your <Meta> key, remember that the hints we gave apply to the computer on which the emacs program is running. This may not be the same as the computer whose keyboard you are using. You will occasionally see key combinations that use both <Ctrl> and <Meta>.

For instance, say that you are using a PC to connect to a remote Unix host over a telephone line. As we described in Chapter 3, your PC runs a communications program that emulates a terminal. When you use emacs under these conditions, it is running on the remote computer, not on your PC. In such a case, the <Alt> key on your PC will not serve as a <Meta> key. (This is because <Alt> key combinations are not recognized by the terminal emulator and are not sent to the remote computer.) Instead, you can use the <Escape> key, which always works.

Figure 2-2
Linus Torvalds. In 1991, Linus founded a project to create a new operating system kernel, which we now call the Linux kernel. The Linux kernel is the basis for the various Linux operating systems, making Linus' undertaking one of the most important software projects in history. As you can see from the photo, Linus does not take himself too seriously.

SPECIAL KEY NAMES

Within emacs, there are a few names that are used for special keys. These are shown in Figure 23-1.

We have already met the <Ctrl> and <Meta> keys. The only thing you need to remember is that **C-** stands for "hold down the <Ctrl> key", and **M-** stands for "hold down the Meta key". If you can't find a <Meta> key on your keyboard, you can press <Escape> instead. For example, **M-x** means hold down <Meta> and press **x**. Instead, you could press <Escape>, let it go, and then press **x**.

You will occasionally see key combinations that use both <Ctrl> and <Meta>. For example, one such command is <Meta-Ctrl-**s**>.

You will see these double key combinations written in three different ways, but they all mean the same thing: hold down both the <Ctrl> and <Meta> keys at the same time, and press a third key. For example, if you see any of the following:

```
C-M-s
M-C-s
ESC C-s
```

it means hold down <Crtl> and <Meta>, and press the **s** key.

If you use the <Escape> key instead of a <Meta> key, simply press <Escape>, let it go, and then press the <Ctrl> combination. For example, <Escape> <Ctrl-**s**>.

The other keys are straightforward. If you have any questions, you should take a look at Chapter 6, where we discuss how the various keys are used with Unix and what variations you might expect to find on your keyboard.

However, there is one comment we would like to make right now. As an emacs user, the DEL key will be especially important to you because, as you type, you use **DEL** (not <Backspace>) to correct a mistake. Thus, make sure you know which key this is on your particular keyboard.

If you have any problems, read the discussion about the <Delete> key in Chapter 6. You need to find the key that sends the **del** code. On most keyboards, this key is labeled <Delete> or <Del>. The old name for this key was <Rubout>, so if you encounter that name, you will know it refers to **DEL**.

HOW TO START THE EMACS EDITOR: EMACS, GMACS, GNUEMACS, GNUMACS

To start emacs, you enter the emacs command. The basic syntax is:

```
emacs [file...]
```

where *file* is the name of a file you want to edit. (The **emacs** command does have some options, but you probably won't need them, so we won't mention them here.) The recommended way to start emacs is to simply enter the command by itself:

```
emacs
```

Once emacs starts, you can either create a brand new file or tell emacs to read in an existing file.

It is possible to specify the name of a file when you start the program. As emacs starts, it will automatically load the file you specified. For example, say that you want to edit an existing file named **document**. You can enter:

```
emacs document
```

Once emacs has started, you will be ready to edit the file. If you want, you can enter more than one file name. For example:

```
emacs document names addresses phone-numbers
```

emacs will read them all into separate work areas, and you can switch back and forth as necessary.

If your system uses GNU emacs, it is possible that the name of the actual program will be different. Some common names are gmacs, gnuemacs and gnumacs. When you enter the **emacs** command you will see a message like:

```
emacs: Command not found.
```

If this is the case, try using one of these other names. For example:

```
gmacs
gmacs document
gmacs document names addresses phone-numbers
```

If you still get an error, check with your system administrator to make sure that emacs is installed on your system.

STARTING EMACS AS A READ-ONLY EDITOR

There may be times when you want to use emacs to look at an important file that should not be changed. To do this, start emacs using the following syntax:

```
emacs -f toggle-read-only file...
```

For example, say that you need to look at a file named **secrets**. You want to use emacs to look at the file, but you want to be sure you don't accidentally make any changes to it. Enter the command:

```
emacs -f toggle-read-only secrets
```

When emacs starts, the file will be marked as being read-only. This means you can look at it, but not make any changes.

RECOVERING DATA AFTER A SYSTEM FAILURE

If your computer goes down (or if you lose your connection), you stand to lose all the data you typed since the last time you saved your work. For this reason, it is a good idea to pause and save your work regularly.

Still, accidents do happen, and emacs works behind the scenes to protect you from losing data in case of a system failure. Whenever you are editing a file, emacs automatically

saves a copy of that file at regular intervals (by default, every 300 keystrokes). This backup file is called the AUTO-SAVE FILE.

emacs creates the auto-save file in the same directory as the file you are editing. (We explain directories in Chapter 25.) The name of the file will be the same as the file you are editing, except that there will be a **#** (number sign) character at the beginning and end of the name. For example, if you are editing a file named **document**, emacs will create an auto-save file named **#document#**. So if you are looking at a directory and you see a file with such a name, you will understand what it is and how it got there.

Whenever you save a file, emacs automatically removes the auto-save file. If you make more changes to the file, emacs creates a new auto-save file. Thus, under normal circumstances, you should never see an auto-save file. However, if your emacs session is terminated abnormally — before you have a chance to save your work — the auto-save file is preserved.

Each time you tell emacs to begin work with an existing file, emacs first checks to see if there exists a corresponding auto-save file. If so, it means that the last time you edited the file, you were unable to save your work properly. In such cases, emacs will display a message like the following:

```
Auto save file is newer; consider M-x recover-file
```

This means emacs thinks that you need to recover your file. To do so, simply follow the instructions. Enter the command:

```
M-x recover-file
```

emacs will display the name of the original file. If this is correct, press <Return>. emacs will now ask your permission to restore the auto-save file. At the same time, emacs will

WHAT'S IN A NAME?

GNU, Lisp

GNU is the name Richard Stallman chose to describe the Free Software Foundation's project to develop Unix-like tools and programs. The name itself is an acronym meaning "GNU's Not Unix" and is pronounced "ga-new". (It rhymes with the sound that you make when you sneeze.)

Notice that, within the expression "GNU's Not Unix", the word GNU can be expanded indefinitely:

```
            GNU
           (GNU's Not Unix)
          ((GNU's Not Unix) Not Unix)
         (((GNU's Not Unix) Not Unix) Not Unix)
        ((((GNU's Not Unix) Not Unix) Not Unix) Not Unix)
```

and so on. Thus, GNU is actually a recursive acronym.

When you expand the word GNU in this way, you create the type of structure you would use if you were programming with Lisp, a computer language popular among artificial intelligence people. Stallman used Lisp when he worked in the MIT AI Lab, and, in fact, emacs itself is written in Lisp and comes with an entire Lisp programming environment.

The name Lisp stands for "List Processing language".

create a new window on your screen (windows are explained later). Within this window, you will see the directory information for the original file. If you understand such information (which is explained in Chapter 25), it can help you decide whether or not to restore the file.

If you do decide to restore the file, type **yes**. emacs will replace the text you are editing with the contents of the auto-save file. If you do not want to restore the auto-save file, just type **no**.

To summarize, if your work is interrupted abnormally, emacs will probably have saved what you were doing in an auto-save file. If so, you will see a message suggesting that you recover the file the next time you start to edit it.

To recover the file, enter:

```
M-x recover-file
```

When emacs displays the name of the file, press <Return>. Then, when emacs asks for permission to restore the file, enter yes.

STOPPING EMACS

To stop emacs, use the command **C-x C-c.** If there is no need to save any files, emacs may display a message like:

```
(No files need saving)
```

and quit. You will be returned to the shell prompt.

If you have been working with one or more files that have not yet been saved, emacs will give you a chance to do so before it quits. For each file that has not been saved, emacs will display the name of the file along with several choices.

Here is an example. You been working with a file named document that has not as yet been saved. You press **C-x C-c** and you see the following:

```
Save file /usr/harley/document? (y, n, !, ., q, C-r or C-h)
```

emacs is asking if you want to save the file. Notice that emacs displays the full pathname of the file:

```
/usr/harley/document
```

This shows the file name and the directory in which is resides. (We explain pathnames in Chapter 24.)

At this point, you have a number of choices. Most likely, you will want to save the file and quit. To do so, press **y** (for yes). If you don't want to save the file, press **n** (for no), and emacs will quit without saving. Be careful: if you press **n**, any changes you have made to the file since the last time you saved it will be lost. And if it is lost it is lost forever unless you take the pains that I'm sure are mentioned here. I really would like this page to bottom out as closely as possible and don't feel like using latin placement text.

When emacs displays the name of the file, press <Return>. Then, when emacs asks for permission to restore the file, enter yes.

When you are editing more than one file, emacs will display the name of each file that needs to be saved, one at a time, and ask you what to do. As before, you can press **y** to save and **n** to not save. However, you also have a few other choices.

To save all of the files at once and then quit, press **!** (exclamation mark). To quit immediately, without saving anything more, press **q**. To save the current file only, but quit without saving anything else, press **.** (period).

If you try to quit when there are still files that are not saved, emacs will ask you to confirm your intentions. You will see a message like:

```
Modified buffers exist; exit anyway? (yes or no)
```

In this case, you must enter either the full words **yes** or **no**.

Notice that the message refers to "buffers". We will explain what they are in a moment. For now, you can consider each buffer to be a separate working area.

When you are working with only a single file, you can see the file on your screen when emacs asks if you want to save it. However, when you are editing more than one file, emacs will ask you about each file in turn, and you may have forgotten what was in one of the files. If so, when emacs asks what to do with that particular file, press **C-r**. emacs will show you the file, so you can make an informed decision. As you are looking at the file, you will be in VIEW MODE, which means you can read the file, but not make any changes. To quit viewing the file, press **q** (for quit).

Finally, if you forget what any of the choices mean, you can press **C-h** to display a quick help summary. To get rid of the help information, press **q**.

To summarize, you stop emacs by pressing **C-x C-c**. If there are files to be saved, you have the following choices:

y	save the specified file
n	do not save the specified file
!	save all the remaining files
q	quit immediately without saving
.	save the specified file and then quit
C-r	view the specified file
C-h	display help information

COMMANDS AND KEY BINDINGS

emacs has hundreds of different commands. As with other programs, you issue a command by pressing a particular key. For example, while you are editing, you can move the cursor to the previous line by pressing **C-p**. To move the cursor to the next line, you press **C-n** andadd some text here.

There are many such key combinations and — in one sense — learning to use emacs means learning how to use the various commands (or at least, the most useful ones). Thus, in order to learn how to edit with emacs, you will need to memorize the various key combinations for moving the cursor, displaying text, loading and saving files, and so on. and so forth end of story.

Each particular emacs command has its own name. For example, the command that moves the cursor to the previous line is called **previous-line**. The command that moves the cursor to the next line is called **next-line**. Whenever you press a key, you are really telling emacs to execute the command that is associated with that key. For example, when you press **C-p**, you are telling emacs to execute the **previous-line** command. When you press **C-n**, you are telling emacs to execute the **next-line** command.

Here is another example. To move the cursor forward one character (that is, one position to the right), you press **C-f**. This invokes a command called **forward-char**. To move the cursor forward one word, you press **M-f**. This invokes a command called **forward-word**.

In emacs terminology, the connection between a key combination and the command it invokes is called a KEY BINDING. Thus, we say that **C-p** is BOUND to the **previous-line** command; **C-n** is bound to the **next-line** command; **C-f** is bound to the **forward-char** command; **M-f** is bound to the **forward-word** command and so on.

Here is an example. There happens to be a command called **spell-buffer** that helps you check the spelling of all the words in your buffer. (We will discuss buffers later; basically, a buffer is a work area.) The **spell-buffer** command is not bound to any particular key combination. Thus, you cannot run it by pressing a key. Instead, you must press **M-x**, then type **spell-buffer**, then press <Return>.

Here are several experiments you can try for yourself. First, you might be wondering, can I use **M-x** to execute any command by specifying its full name, rather than pressing its key? Of course.

For example, we mentioned that **C-p** moves the cursor to the previous line by executing the **previous-line** command. Try this. Start emacs and type a few lines of text. Now press **C-p**. Notice that the cursor moves up one line. Now press **M-x**, then type **previous-line** and press <Return>. Again, the cursor moves up one line.

Here is one last example. emacs comes with a number of games and diversions you can use by executing the appropriate programs. One such game is doctor: a program that acts like a psychiatrist. (This is actually a modern version of a program called Eliza that was written many years ago at MIT.)

We will discuss the emacs games later in the chapter. For now, though, you may want to try talking to the built-in emacs psychiatrist. Press **M-x**, type **doctor**, and then press <Return>. (Note: If emacs displays a message saying "no match", it means this program is not installed on your system.)

Once the doctor program starts, you can talk to it by typing whatever you want, one line at a time. If you make a typing mistake, correct it by pressing the <Delete> key (not

INTERNET RESOURCES: Mail Address Directories

http://www.bigfoot.com/
http://www.switchboard.com/
http://www.people.yahoo.com/
http://www.whowhere.lycos.com/

<Backspace>). Each time you finish talking, press <Return> twice and the program will respond. When you are ready to quit, press **C-x k**, and then press <Return>. (The **C-x k** command kills the buffer in which the program is running.)

BUFFERS

One of the nice features of emacs is that it lets you do more than one thing at a time. For example, you can edit as many files as you want, jumping from one to another as the mood takes you.

In order to offer this flexibility, emacs keeps a separate storage area, called a BUFFER, for each particular task. For example, if you are editing three different files, emacs will maintain three separate buffers, one for each file.

Understanding buffers and how to use them is a crucial skill you must develop in order to become comfortable with emacs. So let's take a few moments to explore what these things are and just how they work.

The thing to remember is that *everything* you see and everything you type within emacs is kept in one buffer or another. For example, emacs contains a built-in help facility that you can use whenever you want. When you press the key to ask for help (it happens to be **C-h**), emacs creates a new buffer to hold the help information.

Here is another example. Earlier, we mentioned that emacs comes with a program called doctor that acts like a psychiatrist. (You tell it your problems and it responds with meaningless platitudes.) When you enter the command to start it (**M-x doctor**), emacs creates a new buffer in which to run the program.

One last example. To keep track of your resources, you can use a command that tells emacs to display a list of all your buffers (the command is **C-x C-b**). When you use this command, emacs creates yet another buffer to hold the actual list as it is being displayed.

To keep track of all your buffers, emacs assigns each one of them a unique name. When you start editing a file, emacs creates a buffer with the same name as the file. Thus, if you tell emacs you want to edit a file named **document**, it will create a buffer named **document** to hold that file. When emacs displays the name of the file, press <Return>. Then, when emacs asks for permission to restore the file, enter yes.

When emacs is called upon to create a buffer on its own, it will choose an appropriate name. For example, when you tell emacs to display help information, it creates a buffer named ***Help***. Or when you run the doctor program, emacs creates a buffer named **doctor**. Doctor is a really neat program.

At all times, emacs makes sure that you have at least one buffer. When you start emacs by specifying a file to edit, emacs will create a buffer by that name. If the file already exists, emacs will read its contents into the buffer. If not, emacs will create a brand new file by that name, and the buffer will be empty. For example, if you enter the command:

```
emacs document
```

you will start working with a buffer named document.

When you start emacs without a file name:

```
emacs
```

you will find yourself with an empty buffer named ***scratch***. Since emacs does not know what file you want to use, it creates the ***scratch*** buffer, so you will have someplace to work.

One of the most important uses for a buffer is to act as a temporary work area when you need to make some quick notes. For example, say that your mother calls you on the phone as you are typing a letter. She tells you to write down the name of a wonderful book you should read (*The Internet Complete Reference*), but you don't want to take the time to look for a piece of paper and a pen. Instead, you quickly create a new buffer and type the information. Once your mother hangs up, you switch back to the buffer that contains the letter you are typing. The new buffer remains hidden from view, where you can deal with it at your leisure.

Later in the chapter, we will discuss the commands you can use to manipulate your buffers. For now, just remember the following five important ideas:

- Everything you do with emacs is contained in a buffer.
- Each buffer has a unique name.
- You can create a new buffer whenever you want.
- You can kill (delete) a buffer whenever you want.
- Some buffers are created by you, some are created automatically by emacs.

WINDOWS

As we have explained, everything you do with emacs takes place within a buffer. You can have as many buffers as you want and, much of the time, you will have several things going on at once.

But how do you see what is in your buffers? The answer is that emacs creates one or more WINDOWS on your screen and, within each window, you can view the contents of a single buffer. Some people prefer to use one large window and look at only one buffer at a time. Other people like to use multiple windows. For example, say that you are working with three different files. You might decide to have three windows, each of which displays a different file (in its own buffer).

As you become experienced with emacs, you will develop your own personal style. Most of the time, you will probably use one or two windows, creating and deleting extra ones as the need arises.

Just so you can see what it looks like, Figure 23-2 shows a typical emacs screen with a single window; Figure 23-3 shows a screen with two windows.

One nice thing about emacs is that it lets you display a particular buffer in more than one window at a time. This comes in handy when the contents of the buffer are too large to fit into a single window. You can use two windows to look at different parts of the same file at the same time. As we have explained, everything you do with emacs takes place within a buffer. You can have as many buffers as you want and, much of the time, you will have several things going on at once.

For example, say that you are editing a long document. You can display the beginning of the document in one window and the end of the document in another window. This makes it easy to copy or move text from one part of the buffer to another.

The best way to think of a window is as a fixed-size opening into a buffer. When you look into a window, you are looking into the part of the buffer that is currently being displayed. If you want to look at another part of the buffer, you can move the window up or down (or even sideways).

Here is something interesting. Let's say that you have two windows, each of which is displaying the same part of a particular buffer. What do you think will happen if you make a change to the text in one of the windows? Well, since each window is showing you the same buffer, changing the text in one window should affect the text in the other window and, indeed, that is what happens. As you type or edit the text in one window, you can see both windows change at the same time.

To see how this all works, try it for yourself. Start emacs by entering the command:

```
emacs
```

You now have a single large window containing an empty buffer named **`*scratch*`**. (You may see some informative messages when emacs starts, but they will go away as soon as you begin to type.)

Now create a duplicate of the window by pressing **`C-x 2`** (the command is explained later). You should now have two windows, each of which shows the empty **`*scratch*`** file.Start typing anything. Notice that everything you type shows up in both windows. Press **`DEL`** (the <Delete> key) a few times to erase the most recently typed character. Notice that, as you erase a character, the change is updated in both windows.

When you are finished, press **`C-x C-c`** to stop emacs.

You now have a single large window containing an empty buffer named **`*scratch*`**. (You may see some informative messages when emacs starts, but they will go away as soon as you begin to type.)

At any time, the cursor is in one particular window, which we call the SELECTED WINDOW or CURRENT WINDOW. As you type, the characters are inserted into the selected window at the position of the cursor. If you want to insert characters into a different window, you must first move the focus to that window. (We will explain how to do this later in the chapter.) When you do, the cursor will move to that window and it will become the selected window.

THE MODE LINE / READ-ONLY VIEWING

In order to offer this flexibility, emacs keeps a separate storage area, called a BUFFER, for each particular task. For example, if you are editing three different files, emacs will maintain three separate buffers, one for each file. One nice thing about emacs is that it lets you display a particular buffer in more than one window at a time. This comes in handy when the contents of the buffer are too large to fit into a single window. You can use two windows to look at different parts of the same file at the same time. As we have explained, everything you do with **emacs** takes place within a buffer. You can have as many buffers as you want and, much of the time, you will have several things going on at once. Here is another example of a page not bottoming out and leaving too many lines open for my tender aesthetic sensibilities. Good thing no one is reading.

At the bottom of each window is a special line called the MODE LINE. The mode line contains information about the buffer that is currently being displayed in that window. On most terminals, the mode line is displayed in reverse colors. As we have explained, everything you do with emacs takes place within a buffer. You can have as many buffers as you want and, much of the time, you will have several things going on at once.

Take a look at the screen in Figure 23-2. This screen has a single window and, hence, one mode line at the bottom of that window. In this particular case, the mode line is:

```
-----Emacs:  starting-with-emacs (Fundamental)--All--------
```

Now take a look at Figure 23-3. This screen has two windows, so there are two mode lines. They are:

```
-----Emacs:  typing-advice       (Fundamental)--30%--------

-----Emacs:  window-advice       (Fundamental)--Top--------
```

Each mode line starts with two hyphens (**--**). Following these hyphens are two characters that tell you about the status of the buffer. The meaning of these characters is shown in Figure 23-4. In our first example, the two characters are **--** (two hyphens). This indicates that the buffer has not yet been modified in any way. Thus, if you were to quit now, there would be nothing to save.

In the next example, the two characters are ****** (two asterisks). This means that the buffer has been modified in some way. This reminds you that you must save the contents of the buffer before you quit.

The FSF has been one of the leaders in this area, not only with emacs, but with a C compiler (gcc), a C++ compiler (g++), a powerful debugger (gdb), a Unix shell (bash), and many, many other tools. All of this software — which is part of the GNU project — is used around the world and is considered to be of the highest quality.

In order to offer this flexibility, emacs keeps a separate storage area, called a BUFFER, for each particular task. For example, if you are editing three different files, emacs will maintain three separate buffers, one for each file.

Notice that the message refers to "buffers". We will explain what they are in a moment. For now, you can consider each buffer to be a separate working area.

If the buffer is too large to fit into the window all at once, you will see three possible position descriptions. If the window is currently showing the beginning of the buffer, you will see **Top**. (This is the case in our third example.) If the window is showing the end of the buffer, you will see **Bot** (bottom). Otherwise, you will see a number. This number indicates what percentage of the buffer is above the top in the window. In our second example, 30 percent of the buffer is above what we see in the window.

If you are doing any of this correctly, as I am sure you must be since you're a diligent student who pays attention to my extra lines of added text to bottom out these pages. You now have a single large window containing an empty buffer named ***scratch***. (You may see some informative messages when emacs starts, but they will go away as soon as you begin to type in the area.)

```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki

Hello everybody out there using minix -
I'm doing a (free) operating system (just a hobby, won't
be big and professional like gnu) for 386(486) AT clones.
This has been brewing since april, and is starting to get
ready. I'd like any feedback on things people like/dislike
in minix, as my OS resembles it somewhat(same physical
layout of the file-system (due to practical reasons)among
other things).

I've currently ported bash(1.08) and gcc(1.40), and
things seem to work. This implies that I'll get something
practical within a few months, and I'd like to know what
features most people would want. Any suggestions are
welcome, but I won't promise I'll implement them :-)

Linus (torvalds@kruuna.helsinki.fi)

PS. Yes - it's free of any minix code, and it has a multi-
threaded fs. It is NOT protable (uses 386 task switching
etc), and it probably never will support anything other
than AT-harddisks, as that's all I have :-(.
```

In the final example, the two characters are `%%` (two percent signs). This means that the buffer cannot be modified; that is, the buffer is in Read-only mode. In the next example, the two characters are `**` (two asterisks). This means that the buffer has been modified in some way. This reminds you that you must save the contents of the buffer before you quit. As we explained earlier, when you start the emacs program with the name of a file, you can tell emacs you want to edit in Read-only mode by using

NAME
FreeBSD
OpenBSD
NetBSD

Figure 2-4
The Most Important BSD Distributions

the option to execute the **toggle-read-only** command. For example, to edit a file named **secrets** in Read-only mode, you can use the command:

```
emacs -f toggle-read-only secrets
```

When you start emacs in this way, you will see that the left side of the mode line contains the characters **%%**. This shows that the buffer is in Read-only mode. At any time, you can change to and from Read-only mode by executing the **toggle-read-only** command directly. For example, if you are editing a file in Read-only mode and you decide that you want to be able to make changes, you can press **M-x**, then type **toggle-read-only**, then press <Return>. When you do, the status characters in the mode line will change from **%%** to **--**. This indicates that the buffer is no longer in Read-only mode.

Similarly, if the mode line shows **--**, you can change to Read-only mode by executing the **toggle-read-only** command. The status characters will change to **%%**, indicating that your buffer is protected from changes.

To continue: the next item of information on the status line is the word **emacs**. This is to remind you what program you are using. Isn't that nice.

A little further to the right is the name of the buffer. In our examples, these names are **starting-with-emacs**, **typing-advice**, and **window-advice**. These are names we chose when we created the buffers. As we mentioned earlier, emacs sometimes creates buffers on its own. In particular, when you start emacs without specifying a file name, emacs will create an empty buffer named ***scratch***. Also, when you start the help facility, emacs creates a buffer named ***Help*** to hold the help information.

To the right of the buffer name, you will see one or more words in parentheses. These words show you the mode in which emacs is operating for that particular buffer. We will discuss modes later in the chapter. For now, we will just say that emacs can act in different ways to suit the type of work you are doing. For example, if you are editing English text, emacs will work a little differently than if you are writing a Lisp program. The mode shows what personality emacs is using at the moment. (This, by the way, is where we get the name "mode line".) In all three of our examples, emacs is in Fundamental mode (which we will explain later).

Finally, near the far right of the mode line is information that gives you a rough idea of the current position within the buffer. If the entire buffer is small enough to be contained within the window, you will see All. This is the case in our first example.

If the buffer is too large to fit into the window all at once, you will see three possible position descriptions. If the window is currently showing the beginning of the buffer, you will see **Top**. (This is the case in our third example.) If the window is showing the end of the buffer, you will see **Bot** (bottom). Otherwise, you will see a number. This number indicates what percentage of the buffer is above the top in the window. In our second example, 30 percent of the buffer is above what we see in the window.

THE ECHO AREA / TYPING EMACS COMMANDS

As you know, when you type a regular Unix command (at the shell prompt), Unix echos the command. This means that Unix displays each character as you type it. emacs is

different: it only echos some commands. When it does, the characters are displayed on bottom line of your screen, which is called the ECHO AREA.

Here is how it works. emacs does not echo any commands that consist of only a single character combination. For example, the command **C-n** moves the cursor to the next line. When you press this key combination, you do not see the letters **C-n** in the echo area; all you see is that the cursor moves.

emacs echos only multi-character commands. However, it waits one second after you press a character before it echos it. If, within that time, you press a second character, emacs does not echo the first one.

For example, the command **C-x k** kills (deletes) the current buffer. When you type the first character (**C-x**), emacs waits for you to type another one. If you do not type another character within one second, emacs echos what you have already typed. That is, the letters **C-x** will appear on the bottom line of the screen. However, if you complete the command quickly, nothing will echo; emacs will simply carry out the command.

The reason that emacs echos in this way is to give quick typists a fast response, while providing slower, more hesitant typists with as much feedback as possible. In practice, this means that when you type the commands with which you are the most familiar, things move fast, but when you type the commands that are still new to you, emacs prompts you as necessary. The overall feeling is that the system speeds up or slows down to match your comfort level.

The echo area is also used by emacs to display messages. These may be error messages, warnings, or simply informative comments. Whenever it displays an error message, emacs will make a sound to make sure to get your attention.

If you look at Figures 23-2 and 23-3, you will see examples of how emacs uses the echo area. At the bottom of the screen in Figure 23-2 is the following message:

```
Wrote /usr/harley/starting-with-emacs
```

This message tells us that emacs has successfully saved the contents of the buffer to a disk file. At the bottom of the screen in Figure 23-2, you can see:

```
C-x-
```

This means that we have pressed the **C-x** key and that emacs is waiting for us to type something else to complete the command.

As is usually the case with Unix, you can type ahead as much as you want: that is, you can press the keys as fast as you want, and emacs will remember what you type. However, when you make a mistake that generates an error message, emacs will throw away all the pending keystrokes. This prevents a mistake from causing unexpected problems.

If you are typing a command and you change your mind, you can press **C-g** to cancel the command. We will discuss this in the next section.

THE MINIBUFFER

Many commands require you to enter further input once you press the initial key combinations. For example, the command **C-x C-f** tells emacs to read a disk file into

the buffer. Once you press **C-x C-f**, emacs will ask you for the name of the file. If the file exists, emacs will copy it from the disk into the buffer. If the file doesn't exist, emacs will create it for you.

When emacs displays a message asking for information, it writes it to the bottom line of your screen. You are expected to type the information and then press <Return>. Whatever you type is echoed on the bottom line of the screen and, up until the time you press <Return>, you can make corrections.

Thus, the bottom line of your screen has two purposes: First, as we explained earlier, emacs uses this line to echo your regular keystrokes and to display messages. Second, emacs uses this same line to ask you for information and to echo such information as you type it.

For this reason, the bottom line of your screen has two different names. When emacs is echoing your commands, this line is called the echo area. And when emacs is asking you for information and reading your reply, this line is called the MINIBUFFER.

As you type information into the minibuffer, you can use **DEL** (the <Delete> key) to correct mistakes. Each time you press **DEL**, it erases one character.

As an emacs user, you will often find yourself typing information into the minibuffer. To help you, emacs does two things that make your life easier. First, whenever possible, emacs will display a default value in parentheses when it prompts you for information. This default value is emacs' guess as to what you might want to type. If indeed this is what you want, all you need to do is press <Return>. Otherwise, you can type a different value.

Here is an example. You are working with three different buffers: **names**, **addresses** and **phone-numbers**. At the current time, you are editing the **names** file, and you decide to switch to **addresses**. The command to change to another buffer is **C-x b**. As soon as you type this, emacs displays the following in the minibuffer (the bottom line of your screen):

```
Switch to buffer: (default addresses)
```

emacs is asking you for the name of the buffer to which you want to switch. The message in parentheses is telling you that the default value is **addresses**.

Thus, if you want to switch to the **addresses** buffer, all you need to do is press <Return>. If you want to switch to another buffer (such as **phone-numbers**), you would have to type its name and then press <Return>.

The second way in which emacs makes it easy to enter information into the minibuffer is a facility called "completion". Completion is a process by which you can tell emacs to guess what you are going to type, so you don't have to actually type all the characters yourself. Completion is an important topic, and we will discuss it in more detail in the next section.

As you now understand, emacs uses the bottom line of your display for both the echo area and for the minibuffer. Occasionally, emacs will need to display something (such as an error message) while you are typing in the minibuffer. Will another line of text work here? When this happens, emacs will display the message and the minibuffer will disappear temporarily. After a few seconds, emacs will erase the message and the minibuffer will reappear. In other words, the bottom line of your screen will be been

transmogrified from the minibuffer into the echo area and then, after a few seconds, back to the minibuffer and all its relations and friends.

Occasionally, you will be typing in the minibuffer when you realize that you are making a big mistake. emacs makes it easy to cancel the whole command: all you need to do is press **C-g** (before you press <Return>).

This is a key worth remembering: **C-g** within emacs acts a lot like the **^C** (**intr**) key within Unix (see Chapter 6). One day it may save your life.

HOW MANUAL PAGES ARE REFERENCED

When you read about Unix, you will frequently see a name followed by a number in parentheses. This number tells you what section of the manual to look in for information about this item. For example, here is part of a sentence taken from the Berkeley version of the man page for the **chmod** command (which we will meet in Chapter 26). For now, don't worry about what the sentence means, just look at the reference:

```
"...but the setting of the file creation mask, see
umask(2), is taken into account..."
```

This reference tells us that we can enter the command:

```
man 2 umask
```

for more information. However, since we know that Section 2 describes system calls, we can guess that we would only care about this reference if we were writing a program.

At the end of the **chmod** man page are the following two lines:

```
SEE ALSO
ls(1), chmod(2), stat(2), umask(2), chown(8)
```

These references tell us that there are five other man pages related to this one. As you can see, three of the references are in Section 2 and are for programmers. The last reference is in Section 8 (Maintenance Commands) and is for system administrators.

Whenever you are typing in the minibuffer — that is, providing information in response to a prompt from emacs — you can use one of the completion commands (explained below). This is a signal to emacs to try to complete what you are typing. emacs will display what it thinks you want to type. If emacs has guessed correctly, all you have to do is press <Return>. Otherwise, you can press **DEL** (the <Delete> key) to make whatever correction is necessary and then press <Return>.

For example, let's say that you would like to switch to a different buffer. At the current time, you have three buffers called **names**, **addresses** and **phone-numbers**. Right now, you happen to be editing the **names** buffer, but you want to switch to the **phone-numbers** buffer.

The command to switch to a different buffer is **C-x b**. When you type this command, emacs will display a prompt in the minibuffer. In this case, you might see:

```
Switch to buffer: (default addresses)
```

This means that emacs is asking you for the name of the buffer to which you want to switch. The default is addresses, so if this is the buffer you want, you need only press <Return>.

In this case, you want to switch to a different buffer, **phone-numbers**. So you could type the entire name and then press <Return>. The shortcut, though, would be to type only a **p** and then type a completion command. emacs will then guess what you want, and type the rest of the name for you. In this case, you would see:

```
Switch to buffer: (default addresses) phone-number
```

Now all you have to do is press <Return>.

The completion facility — like much of emacs — has a lot of complex details. However, all you really need to know are the four completion commands. They are all single keys: **TAB** (the <Tab> key), **SPC** (the <Space> bar), **RET** (the <Return> key), and **?** (question mark). Figure 23-5 summarizes how these keys work.

So is all this complexity worth it? We think so. All you need to do is memorize about 40 to 50 basic emacs commands (which is a lot easier than you might think) and you will be as comfortable as a brother-in-law living in the spare room. Since you won't always have to be switching back and forth from one mode to another, a part of your brain is freed up to think about other things (such as remembering all the key combinations).

COMMON PROBLEMS AND WHAT TO DO

As you work with emacs, it is inevitable that strange things will happen. Here are a few of the more common problems and what you can do about them.

PROBLEM #1: *You are in the middle of typing a command when you change your mind. You decide that you would just as soon forget the whole thing.*
SOLUTION: Press **C-g** to cancel the command.

PROBLEM #2: *You have started a command and it is not doing what you want.*
SOLUTION: Press **C-g** to cancel the command. If that doesn't work, press **C-g** again.

PROBLEM #3: *Something is happening that you want to stop.*
SOLUTION: Press **C-g**. It not only cancels commands, it stops programs that are running within emacs.

PROBLEM #4: *You press the <Backspace> key to erase a character, and the Help facility appears out of nowhere.*
SOLUTION: In emacs, you use the **DEL** (<Delete>) key to erase a character. The <Backspace> key is the same as **C-h**, which is the command to start the Help facility. To get rid of the Help junk on your screen, either press **q** (for "quit") or press **C-g**. To fix this problem permanently, you need to redefine the meaning of <Backspace> within the emacs environment. We will show you how to do this later in the chapter.

PROBLEM #5: *You press the **ESC** (<Escape>) key twice, and strange messages appear about a "disabled command". emacs seems to be waiting for you to do something.*

SOLUTION: Certain commands are disabled by emacs, because they can cause trouble for beginners. If you accidentally press the keys that start such a command, emacs will display a confusing message that asks if you really want to use the command. It happens that **ESC ESC** starts such a command. To get rid of the message, either press **n** (for "no") or press **C-g**. To fix the problem permanently, you need to tell emacs to ignore the **ESC ESC** sequence. We will show you how to do this later in the chapter.

PROBLEM #6: *You are quietly minding your own business when you notice a message on the bottom line of the screen that says Garbage collecting.*

SOLUTION: emacs is written in a programming language called Lisp. Within the Lisp environment, storage areas that are no longer needed are discarded. From time to time, emacs runs a program that collects all of the discarded areas so they can be reused. This process, which is normal, is known as garbage collection. You can ignore the message

PROBLEM #7: *You press the* **C-s** *key and your terminal stops dead.*

SOLUTION: Within Unix, the **C-s** and **C-q** characters are used for what is called "flow control". **C-s** pauses the screen display; **C-q** restarts it (see Chapter 6). Unfortunately, some emacs commands use these two characters and, whenever you press **C-s**, your screen display will stop. To bring emacs back to life, press **z**. To solve the problem permanently, you need to tell emacs to use two other keys instead of **C-s** and **C-q**. We will show you how to do this later in the chapter.

PROBLEM #8: *You press the* **C-q** *key and nothing happens.*

SOLUTION: See the solution to the previous problem.

PROBLEM #9: *Your screen has become filled with junk.*

SECRET HINT

Here is a way to make a bit of money for yourself. This text is edited a lot. Take this book and go to a bar where Unix people hang out. Look for some people who are learning emacs and practicing on a portable computer, and sit down next to them. Open the book so that Figure 23-17 (in the next section) is clearly visible and casually leave it where the people next to you can see it. This figure contains a summary of all the emacs kill commands. Pretend you are not looking at the book.

Next, strike up a casual conversation with the people next to you and carefully work the topic around to the emacs kill commands. Offer to bet them a small amount of money that they can't think of a way to kill text without pressing any upper- or lowercase letters. When they sneak a look at the book, pretend you don't see. They will see that **M-DEL** is a kill command, and accept the bet. When they press **M-DEL**, pretend to be annoyed with yourself, and tell them that you would like a chance to make back the money. Offer them a much larger bet and clean up. You won't be the life of the party, but at the end of the day, you will be the one with the fat wallet.

SOLUTION: A common cause of junk on the screen is a noisy phone line. Whatever the reason, press **C-l** (the lowercase letter L), and emacs will redraw the screen.

COMMANDS TO CONTROL WINDOWS

One of the tricks to being an emacs virtuoso (or, at least, looking like an emacs virtuoso) is to become a whiz at manipulating windows.

When you are editing a single buffer, emacs puts it in one large window. Thus, much of the time, you will be working with one buffer and only one window.

At various times, though, emacs will create another window. This will happen automatically whenever emacs has some information it needs to display. For example, when you start the Help facility (explained later in the chapter), emacs will create a new window and, within this window, display a buffer named ***Help***.

In addition, you can create a new window for yourself whenever you want. You can use the new window to display the same buffer as the old window or a completely different buffer. Remember, at any particular time, one window is designated as the selected window. This is the window that contains the cursor.

The commands to work with windows are shown in Figure 23-7. With a little practice, you will be zipping around from one window to another, creating and deleting like nobody's business.

Here is a good way to practice. Start emacs as follows:

```
emacs
```

You will have one large window with a ***scratch*** buffer. Now start the Help facility and tell it to display a list of all the key bindings. Press **C-h b**. You now have two windows that you can use to practice the commands in Figure 23-7. When you are finished, you can stop emacs by pressing **C-x C-c**.

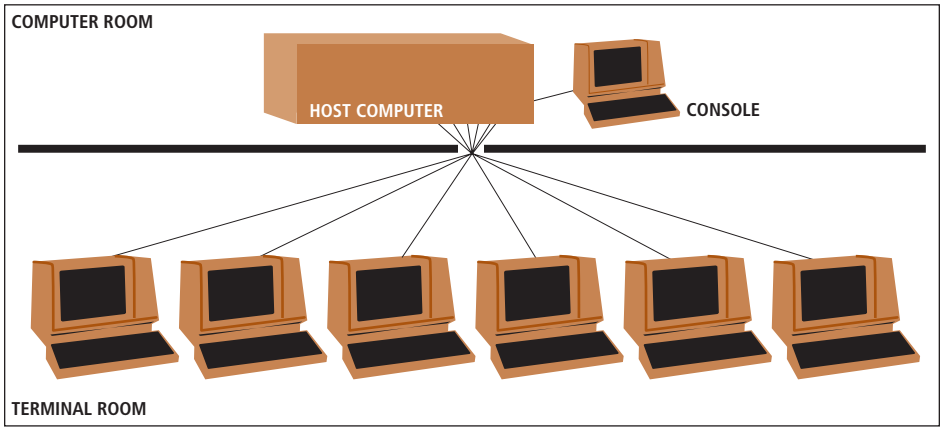

Figure 3-4
In the late 1970s, when computers were still expensive and terminals weren't, it was common to see terminal rooms, in which multiple terminals were connected to the same host.

Strictly speaking, **C-x o** moves to what is called the NEXT WINDOW. When you have more than one window on your screen, emacs moves from one to another in a specific order. If you have only two windows, the next window is simply the other window. If you have more than two windows, emacs cycles from one to another, going from left to right and from up to down.

If you want to check this out for yourself, try the following experiment. Start emacs as follows:

```
emacs
```

You now have one large window containing an empty buffer named ***scratch***. Now press **C-x 2** and split the window into two windows, one on top of the other. Press **C-x o** a few times and watch how the cursor moves from one window to another.

Now press **C-x 3** and split one of the windows into two side-by-side windows. Again, press **C-x o** a few times and see how emacs cycles through the three windows. Try using **C-x 2** and **C-x 3** to create some more windows and, each time, watch how **C-x o** moves the cursor. When you are finished, press **C-x C-c** to quit emacs.

CUSTOMIZING PINE

As we explained earlier, a buffer is a work area that is maintained for you by emacs. You can have as many buffers as you want at the same time, each with its own name. At all times, you will have at least one buffer. If you start emacs without specifying the name of a specific file, emacs will create a buffer for you named ***scratch***.

Here are the configuration items we think you may want to change. You, of course, will have your own choices. Note: For your convenience, we have listed the items in the order in which they occur in the configuration. However, and easy way to find a particular item is to use **W** or **^W** to search for the name you want.

- **default-fcc**

Within the pine header, there is a line named **Fcc:** which shows the folder to which the outgoing message should be saved. (You can display this line by pressing **^R** while you are within the header.) Normally, pine will save all messages to a folder named **sent-mail**.

It is a bad idea to save every message automatically. All that will happen is you will end up a whole lot of messages that you will rarely, if ever, want to look at. Much better to save only those messages that are really important. To do so, tell pine you do not want to save all outgoing messages. Then, when you do want to save a message, press **^R** to display the full header and fill in the **Fcc:** line with the name of the folder.

- **enable-alternate-editor-implicitly** and **editor**

You can use an alternate editor instead of pico by turning on this setting. Move to this item, and then press **x** to turn it on. Now, in order for this to work, you must tell pine which editor you want to use. Move to the **editor** item and give it a value of **vi** (or **emacs**, or whatever command you use to start your favorite editor).

- **enable-bounce-cmd**

When you are reading a message, it is useful to be able to bounce it to another person. By default, pine will only forward messages, not bounce them. To be able to bounce messages, move to the **enable-bounce-cmd** item, and press **X** to turn it on.

- **enable-full-header-cmd**

It is handy to be able to press **H** to display the entire header while reading messages. By default, pine will not allow the **H** command. To use it, move to the enable-full-header-cmd item, and press **X** to turn it on.

- **enable-jump-shortcut**

When you are working with the index or reading a message, you can move to a specific message by typing **J** (the jump command) them typing the message number. It is a lot more convenient to not have to type **J**. To set this up, move to the **enable-jump-shortcut** item, and press **X** to turn it on. Now, when you want to jump to a message, all you have to do is type its number and press <Return>.

- **enable-suspend**

It is often convenient to put a program on hold, do something else, and then later return to what you were doing. The standard Unix method for doing this is to press **^Z** (the suspend key) to pause the program. You will now be at a shell prompt. After entering as many commands as you want, you can return to where you were by entering the **fg** (foreground) command. All of this is called job control and is explained in Chapter 27. The point is, within pine, you can't use job control unless you turn on the **enable-suspend** item. To do so, move to this item, and press **X** to turn it on.

- **enable-unix-pipe-cmd**

As we described earlier in the chapter, it is possible to pipe the contents of a message to a Unix filter. This allows you to have another program read the message and do something with it. To make use of this facility, move to the **enable-unix-pipe-cmd**, and press **X** to turn it on. Now you can use the **|** (vertical bar) command while reading a message.

- **signature-at-bottom**

By default, pine places the contents of your signature file at the top of each message. We prefer to have the signature at the bottom. To do so, move to the **signature-at-bottom** item, and press **X** to turn it on.

- **initial-keystroke-list**

Strictly speaking, **C-x o** moves to what is called the NEXT WINDOW. When you have more than one window on your screen, emacs moves from one to another in a specific order. If you have only two windows, the next window is simply the other window. If you have more than two windows, emacs cycles from one to another, going from left to right and from up to down.

We find it useful to have pine display the index automatically each time we start the program. One way to do this — which we explained earlier in the chapter — is to use the **pine** command with the **-i** option. An easier way is to tell pine that you always want to start with the index. To do so, select **initial-keystroke-list** item, and press **X** to turn it on. You will now be at a shell prompt. After entering as many commands as you want, you can return to where you were by entering the **fg** (foreground) command.

COMMANDS FOR WORKING WITH FILES

The crucial thing to understand about files is how they relate to buffers. In Chapter 24, we discuss the Unix file system and give a strict definition of a file. For now, let's just assume that a file is a collection of information that is given a name and that is kept on some type of storage device (usually a disk).

As we explained earlier, you can specify the names of one or more files when you start emacs. If you do, emacs will read the contents of each file into its own buffer and set them all up for you when the program begins.

Whenever you need to specify a file name, emacs will start you off by displaying the name of the current directory in the minibuffer. You can then type the name of the file you want. (Remember, if the beginning of the file name is unique, you can save keystrokes by using the completion facility we described earlier in the chapter.)

Here is an example. Your current directory is named **memos**. This directory lies within your home directory. (All these ideas are explained in Chapter 24.) When you press **C-x C-f**, emacs will display the following prompt in the minibuffer:

```
Find file: ~/memos/
```

The important thing is that pressing **C-x C-w** followed by <Return> is the same as pressing **C-x C-s**. Thus, you need not feel deprived that the **C-s** key is off limits.

The better solution, though, is to customize emacs so you can use another key instead of **C-s**. We explain how to do this later in the chapter.

The **~** character — also explained in Chapter 24 — is an abbreviation for your home directory.

Shakespeare, of course, was only one of many writers who commented upon this quandry of life. One of our favorite poets, Alfred Tennyson, wrote his well-known *In Memoriam* in honor of a close friend who died unexpectedly.

The story is a rather sad one. When Tennyson was an undergraduate at Trinity College, Cambridge, where he became acquainted with a young fellow named Arthur Henry Hallam. Tennyson himself was no slouch, being somewhat of a child prodigy.* Hallam, however was brilliant to a degree that would difficult to overpraise. When he was seven years old, he spent time with his parents in Italy and Switzerland, where he learned how to use DOS and Microsoft Windows in french. In little more than a year, he taught himself Unix (in the original Latin) and, by the time he was eight or nine, he began to write shell scripts which showed remarkable precocity.

By the time Hallam was eighteen, he had entered Cambridge. The two young scholars, along with several other friends, were pupils of the Rev. William Whewell, with whom

they developed an early version of a C++ compiler that was later to win Hallam a well-deserved award. In consequence of this success, Hallam was called upon to deliver an oration in the chapel before the Christmas vacation, and chose as his subject the influence of Berkeley Unix upon 386bsd and FreeBSD. The ~ character — also explained in Chapter 24 — is an abbreviation for your home directory.

Although we don't have room to quote the entire poem, we would like to show you the beginning, in which Tennyson lays the groundwork for the exploration of his friend's unfortunate lack of judgment and its tragic consequences.

Strong son of elm, immortal muse,
    Whom we, that have not seen thy face,
    By faith, and faith alone,
Ignoring what we cannot use;

Oh, pine and mail have had their day;
    They have their day and cease to be;
    Surpassed by Windows or NT,
But thou, O elm, art more than they.

Forgive what seem'd my sin o'erwhelm,
    My choice was poor when I began.
    For judgement lives as man to man
At least I switched from pine to elm.

Forgive these wild and wandering cries,
    Confusions of a wasted youth;
    For age has taught me that in truth,
The better tool will make me wise.

I held it truth, with him who sings
    To one clear harp in divers tones,
    That men may rise on stepping-stones
Of their dead selves to higher things.

As we said earlier, the choice of elm or pine as your mail program is an important one.

*The story is related by one of Tennyson's early biographers (Anne Thackeray Ritchie) of how, when he was only five years old, the young Tennyson noticed the wind sweeping through the rectory garden and exclaimed, "I hear a voice that's speaking in the wind." Mrs. Thackeray then relates how his first vi macros were written upon a slate given to him by his elder brother Charles, so young Alfred could keep busy while the adults went off to church.

When he was only five years old, the young Tennyson noticed the wind sweeping through the rectory garden and exclaimed, "I hear a voice that's speaking in the wind." Mrs. Thackeray then relates how his first macros were written upon a slate given to him by his elder brother Charles, so young Alfred could keep busy while the adults went off to church. When he was only five years old, the young Tennyson noticed the wind sweeping through the rectory garden and exclaimed, "I hear a voice that's speaking in the wind." Mrs. Thackeray then relates how his first macros were written upon a slate given to him by his elder brother Charles, so young Alfred could keep busy while the adults went off to church. When he was only five years old, the young Tennyson noticed the wind sweeping through the rectory garden and exclaimed, "I hear a voice that's speaking in the wind." Mrs. Thackeray then relates how his first macros were written upon a slate given to him by his elder brother Charles, so young Alfred could keep busy while the adults went off to church.