by Peter Collinson, **Hillside Systems**

CHRIS FAUST

Editing with Emacs

While scanning through the August edition of *Server/Workstation Expert*, I noticed that the Jeffs admitted to switching to `emacs` rather than using `vi`. The choice of a text editor is essentially a religious issue, but we were not told whether the Jeffs were now converted. Of course, debating the advantages of one over another happens frequently and can often lead to overheated arguments that have no real conclusion. It's really a good idea to make up your own mind about the issues, and, of course, as a UNIX user, you don't have to let the machine control you as do the users of the other significant operating systems in the world. You have a choice.

The editor you choose depends on the usage you make of the machine. UNIX users need to generate or modify text files at some point or other. But if you do it rarely, your system has a simple notepad style editor that interacts with your windowing system allowing you to load some text and poke at it with the keyboard and mouse. Examples of these are: CDE's `dtpad` editor that's usually found by clicking the Text Editor menu selection on your desktop; the older `textedit` program supplied as part of Open Windows on Suns; or GNOME's `Edit` program.

If you are used to these simple programs, you may find that `emacs` is easier to learn than `vi`. I find this hard to judge because I've been at the editing game for so long. Of course, both editors were developed in the days before the mouse was invented and were designed to be used on the cursor-addressable terminals from the 1980s. Both had the same problem to solve–the user had only a keyboard for input and needed to be able to enter text and also needed to control the editing functions of the editor using the same set of keys.

Standard ASCII keyboards have a "Control" key that is used in a chording manner with the other keys and generates a special code the computer can recognize. Holding down "A" generates one code value, holding down "Control" and "A" modifies that value. The Control key was there originally to allow teletype users to punch line control sequences onto paper tape. When used in combination with each of the other keys, the Control Key gives the terminal user a set of two-finger keystrokes that don't show on the screen but can be recognized by the program and used as functions for the editor. We often call this set of keystrokes "the Control Keys." However, the set of control keys are limited in number and you soon run out of them. Editors have more functions than there are keys.

The `vi` editor sprang from `ed` and was natural to solve the problem of lack of keys by using "modes." The program starts up in "command mode" and all characters delivered from the keyboard are understood to be editor commands that control the editor, perhaps positioning the cursor in the file being edited. Characters can also be typed in sequence to generate more complex commands. When the user types a command that

places new text in the file being edited, vi enters "input mode." In input mode, characters supplied by the keyboard are no longer interpreted as commands but as characters to be entered into the file. The Escape key is used to move back from input mode into command mode. The ed editor operates in a very similar fashion, although the action is not WYSIWYG.

Because these editors are modal, a key press like "A" means different things depending on the current input state of the program. At one point the keystroke is telling the editor to perform an editing action; at another it is a character to be added into the file. This overloading of key strokes can be confusing. Early versions of vi didn't tell you which mode it was in at any time, and I certainly became used to pressing Escape until the terminal bell rang, ensuring that the editor was in command mode. Users of vi will say that the modal nature of the editor is no problem to them, and its action works well. It's one of those points for religious discussion I talked about above.

The emacs family of editors are "modeless." When you start editing a file and type a printable character, that character is entered into the buffer. Keystrokes do not change meaning depending on the context in which you use them, unless you ask emacs to do that for you. Control keys are used to move about the buffer, and when you run out of those, the control keys have to be sent in sequences that perform more complex actions.

When emacs was originally conceived, the terminals at MIT had an additional control key, the Meta key, that worked in a similar way to the Control key. The Meta key could be used to generate an additional set of control commands when combined with the normal keyboard. Later terminals did not have this ability, and emacs adopted the use of the Escape key as the first character of a command sequence. So where you used to chord the Meta key and say "a" to generate a command on the original systems, you now have to type "Escape" then "a." Emacs documentation refers to this command sequence as M-a, for Meta-a. As time has progressed, emacs has acquired other command sequence introduction characters, such as Control-X and Control-C, which are commonly used today. On my Sun keyboard, the key with a small diamond on it operates like the original Meta key, although I rarely use this feature because it doesn't work everywhere and it's a bad idea to teach your body non-portable habits.

Why I Switched to *Emacs*

I was drawn to emacs originally because I realized my work would be enhanced if I could see more than one file at any one time. The legacy of ed meant that vi was then limited to editing a single file, although you could edit files in sequence and move between them. I found myself jumping from one file to another to find information or pick up text to be pasted into the file I was working on. When dealing with large programs comprising several files, I'd spend time looking up details and then forget them while getting back to the file that I was editing.

Emacs offers the ability to see more than a single file at any one time by splitting the screen horizontally (and vertically if you are so minded). You can also have several views into the same file, and I make considerable use of this ability. At the time I switched to emacs, I was using a glass terminal and my work patterns changed immensely. I stopped using paper for listings and used the editor to view files with the consequent ability to search for information and find it quickly.

Like other editors, emacs makes a copy of the file you are editing, you make changes to the copy and write it back to disk when you are done. Emacs calls this copy a "buffer," and allows you to edit multiple buffers at any one time. By default, emacs provides for automatic backup of files that you change, and will leave files ending in the tilde character (~) all over the file system. I generally arrange to delete these nightly if they are older than a couple of days.

Splitting the screen and loading a new file into a portion takes very few keystrokes, and I still find that I do this when working with text rather than mess around with the mouse to generate a new window with a new invocation of the editor. Actually, when running emacs under the X Window System, it can generate a new window (which it calls a "frame"), and I do use this ability occasionally.

At the bottom of each displayed buffer is a status line that tells you things about the file being edited, and is used to split up the screen. Right at the bottom of each frame underneath the bottom status line is a single line that's the "minibuffer." The minibuffer is used to display messages that either help or complain that you've done something wrong. It is also used as an input area to allow you to add additional information into commands. For example, when you want to search through a file, you'll use the menu or the relevant keystrokes to start the search and type the search string into the minibuffer.

The second feature I realized that I disliked in vi was its treatment of text lines. Again, the legacy of ed meant that vi is really an editor that works on lines. If you are viewing a file, and hold down the key that moves forward one character, then the cursor will "stick" at the end of a line. You then need to use another keystroke to move down to the next line. Emacs treats lines like they are connected together and you can move from one end of the file to another by holding down a single key. I find that I make so much use of this feature that I routinely increase the repeat speed of characters from the keyboard to make it happen much more quickly. This may seem a really trivial reason for changing from one editor to another, but as I said at the start, editor choice is a religious issue.

Using *Emacs*

I don't intend to go into great detail about what keystrokes do what, because emacs does that itself reasonably well (check out the online tutorial). However, it's a good idea to get a handle on the way that commands are organized so that you can understand the documentation and have some idea on how to get help. Of course, emacs is a very mature program and expedience has sometime got in the way of the organization.

You will need to learn the basic default movement keys, and actually this is no bad thing because the key sequences are used widely in other UNIX applications. The Korn Shell, bash and every X type-in box support the emacs keystrokes allowing you to edit input into GUIs. They will also work in some of the simple text editors I mentioned at the start of the article.

Primary file movement in emacs is made with control keys that are somewhat mnemonic. This diagram is stolen from the emacs documentation (see Figure 1).

The C-n notation means "hold down the Control key and the N key." So we have "Control-b" for backward one character, "Control-f" for forward one character and so on. Control keys are mostly used for movement around the file, but they do have other functions. It's useful to know that C-g is used to say "stop what you are doing and get me back to input mode." However, we soon run out of the control keys for actions, and as I said above, emacs uses several command characters as sequence introducers, and also supports the Meta key notion.

The key choices for Meta sequences often map onto the key choices for the control characters but do something that involves a larger movement. For example, C-f moves forward one character, and M-f moves forward one word. C-a moves to the beginning of a line, M-a moves to the beginning of the sentence. C-e moves to the end of a line, M-e moves to the end of a sentence. Sometimes the Meta sequence does the inverse operation, so C-v scrolls down one screenfull (moving the data up), and M-v scrolls up one screenfull (moving the data down).

Emacs has several other standard "introducer" characters that are used to create other classes of commands. C-x followed by another character mostly provides buffer and frame management commands. It's always useful to know that the C-x C-c sequence will get you out of the editor. C-c is also used as an "application" prefix, permitting applications within emacs to introduce their own keys for their own purposes.

C-h is important for novices and experienced users. It allows access to the "Help" functions. They're not too good at telling you how to use editor features, but are good if you have an inkling that something is available and want to know how to use it.

Binding and Customization

Emacs is actually a LISP application. The core of the editor is in C, but many of the functions are written in LISP that is bound into the core either dynamically or at compile time. All keys are "shortcuts" to a set of named functions. When you load the editor, it loads a set of default bindings between the keystrokes and the actions they make. You can also augment the standard bindings by creating your own customization file, called .emacs on your home directory. It's a bad idea for novices to do this, because the editor's behavior will diverge from the documentation and can be a source of confusion.

At any time, you can type M-x. A colon will appear in the minibuffer, and then you need to type a command name. Command names are mostly words separated by hyphens, for example: C-f is usually bound to the command

```
forward-char
```

Emacs will perform command expansion for you in the minibuffer, so you can type some portion of a command, hit space, and emacs will complete the command for you, assuming what you have typed so far is unique. If not, emacs will offer a list of options.

Of course, a command can be bound to several different keystrokes, so if you are comfortable using the arrow keys on your keyboard, you will find that you can move around files using them. Emacs also has bindings for other named keys on my Sun keyboard. Be careful when using these, you will have some difficulty when moving to other keyboards.

There are a great many commands that are not bound to any key in particular and you will find yourself typing

```
M-x command
```

to access them. However, the big win is that you can set up a binding between a keystroke sequence and an action very easily. It is a bad idea to rebind the commonly used keystrokes to something different; it destroys your personal portability to other systems.

The ability to rebind keys has given rise to a huge number of "Modes" inside emacs. For instance, when you are editing a Perl source file, you can enter "Perl-mode" and the same keystrokes will have a different effect. Usually the new binding make sense, so "go to end of sentence" becomes "go to the end of the next function." You can enter a mode automatically depending on the suffix attached to a file, so "Perl-mode" is entered when you are editing a file that ends in .pl, "html-mode" is entered when you are editing a file that ends in .html or .htm, and so on. Modes are used considerably to give context for editing actions, and to permit emacs to do sensible things for you.

For example, if you are editing something that has a regular bracketing and quoting structure, then emacs will track opens and closes for the brackets and quotes. When you type a closing bracket, emacs will briefly highlight the opening bracket so you can see that the correct match has been made. Also, in

Figure 1. Moving the cursor in *emacs*

```
                    Previous line, C-p
                            :
                            :
Backward, C-b .... Current cursor position .... Forward, C-f
                            :
                            :
                       Next line, C-n
```

the modes for programming languages, emacs will provide "correct" indentation automatically depending on your bracket level. I find that Perl's zillions of different bracketing options can sometimes fool the program. Also, I have found myself fighting some aspects of the default customization; for example, I don't like the way that tabs are treated by the defaults.

When using Emacs, I often find that "odd things happen"–usually by hitting a key and finding that it's bound by default to some esoteric command. After a bit you get used to this. Maybe you have found a command that you will use all the time, or you may wish to never use it. In the latter case, you can bind the key to an action that's more useful to you, and relegate the command to an action where you have to type M-x and the command name.

There are several different emacs subsystems that plunge you into a new mode to view their information. Keystrokes are used to control what happens in the modes, and I find that some of these keystrokes are often not well defined. Many of these modes use the space bar to mean "do the next thing," and it's taken me some time to work that out.

It takes some time and experimentation before you are making the program work as it's supposed to. In general, the documentation doesn't often tell you the "thinking" that went on when a feature was designed. For some complex modes, you are never sure if you are using them "properly." Mileage varies.

Nice Features

Since emacs deals with binary files, it can cope with some text files that don't "look" like text files to other UNIX editors. Specifically it can cope with text files with extremely long lines, which is a boon when editing the HTML produced by certain WYSIWYG HTML editors.

I really like the "undo" feature that lets you backtrack through all the edits you have done to get to some known point. This is great in an editor where sometimes I mistype keystrokes intending to do one thing, but obtain some other result.

Emacs has the ability to show different colors and fonts for the data that it's displaying. The font coloring is not stored with the data, but is applied as you edit the file or when it's loaded into a buffer. The coloring is again dependent on the mode of the file. I find it helpful to have the different elements of the source colored differently when programming or writing HTML. For example, in Perl-mode language keywords, string contents, comments, and variable names are all colored differently. The ability to detect that you have not terminated a string saves oodles of time with the Perl compiler.

Finally, emacs can run other programs either in a buffer using a standard shell, or as part of a subsystem interpreting output and allowing you to interact with the program. This ability is used to provide spell checking for text documents (using the ispell program), and for providing Web browsers, mail handlers, news readers and other complex subsystems. All the emacs books tend to concentrate on these features, that apart from spell checking, I rarely use.

Emacs is probably the largest and most complex application that has ever been written for UNIX, and some people use it as their desktop. I personally like to have applications popping up and down on the screen and start programs frequently. I actually tend to use emacs only for editing files. In the early days, when I complained to an emacs user that start-up was so slow, the person would invariably say they started it at the beginning of the day and remained in it all the time. My complaints on performance have stopped as machines have become faster, memories on machines have grown, and speed issues have generally disappeared.

Emacs Versions

The original emacs editor was written by Richard Stallman, the creator of both the GNU project and the Free Software Foundation. Let me quote from the emacs FAQ, that comes with the editor. "The first emacs was a set of macros written in 1976 at MIT by RMS for the editor TECO (Text Editor and COrrector, originally Tape Editor and COrrector) under ITS on a PDP-10. RMS had already extended TECO with a 'real-time' full screen mode with reprogrammable keys. Emacs was started by Guy Steele as a project to unify the many divergent TECO command sets and key bindings at MIT, and completed by RMS." Of course, RMS in the text is shorthand for Stallman.

Since *emacs* deals with binary files, it can cope with some text files that don't "look" like text files to other UNIX editors.

Emacs has now been ported to all the operating systems that are in common use. There are really two extant UNIX versions. GNU emacs, which is supported and distributed by the Free Software Foundation, and XEmacs which started as a split from GNU's source. It is intended to operate in the X Window environment and has some more sexy looking icons. I've never used XEmacs so I cannot comment on the differences between the two.

As I said, emacs is vast, and it pays to "work direct" your knowledge of it. I am always finding some new useful feature that I kick myself for not finding before. Keep reading the documentation.

Further Information

You can get the latest version of emacs from your nearest GNU storage site. It will probably be installed on your machine. I would recommend the O'Reilly book on emacs, *Learning GNU Emacs* by Debra Cameron, Bill Rosenblatt & Eric Raymond. It's now in its second edition, ISBN 1-56592-152-6. It does have some missing bits, which is probably inevitable in a program like emacs. O'Reilly also publishes several other emacs books, so check out the Web site. ⊸

Peter Collinson *runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever … He writes, teaches, consults and programs using Solaris running on an UltraSPARC/10. Email:* pc@cpg.com.