HONEYWELL

# LEVEL 68 INTRODUCTION TO EMACS TEXT EDITOR

SOFTWARE

LEVEL 68

# INTRODUCTION TO EMACS TEXT EDITOR

SUBJECT

Introduction to the Emacs Text Editor and Description of the Most Generally Used Editing Requests of Emacs Fundamental Mode

SPECIAL INSTRUCTIONS

This manual presupposes some basic knowledge of the Multics system provided by the 2-volume set, *New Users' Introduction to Multics*. Some of the preliminary information covered in that set is summarized briefly here, however, so that users at any level of experience can comprehend the techniques presented in this manual.

SOFTWARE SUPPORTED

Multics Software Release 8.2

ORDER NUMBER

CP31-00

March 1981

Honeywell

# *Preface*

This book is an introduction to the Multics Emacs text editor, a real-time editing and formatting system designed for use on video terminals. The Emacs editor has many powerful features and requests not described here. Instead, only the basic requests and those thought to be most helpful for the general user are included. Users desiring a complete description of the Emacs editor should refer to the *Emacs Text Editor Users' Guide,*Order No. CH27. In addition, the *Emacs Extension Writers' Guide,*Order No. CJ52, is available for more advanced users who want to write their own editor extensions.

For many users, however, this manual provides all the information needed. Since the Emacs editor includes a self-documentation feature (described in Section 9), users can easily teach themselves the remaining requests. Users are expected to be familiar with the Multics concepts described in the 2-volume set, *New Users' Introduction to Multics – Part I* (Order No. CH24), and *— Part II* (Order No. CH25), referred to in this book as New Users' Intro.

The term "file" is used interchangeably with "segment" in this manual, since many of the editing requests have the word "file" as part of their command-names. Throughout the manual, the Emacs text editor is frequently referred to as "Emacs"; the Multics Operating System is referred to as "Multics." Technical or other unfamiliar terms are printed as all uppercase when first introduced, and are included in the Glossary (Appendix C).

Section 1 is an introduction to the Emacs editor, the video terminal, logging in, and entering the editor environment. Section 2 expands on the description of the editor and how it works.

Section 3 describes the basic requests for positioning the cursor while editing and describes how to exit from Emacs and log out. Section 4 explains the requests for deleting text and retrieving deleted text. Section 5 shows how to manipulate files (segments) and buffers, and Section 6 describes several requests for formatting text.

Section 7 includes requests for locating character strings and globally substituting one string for another. Section 8 describes some requests that enhance typing convenience. Finally, Section 9 describes the Emacs self-documentation features and how to use them to extend your knowledge of Emacs.

Three appendixes are included as reference aids. Appendix A alphabetically lists the requests described in this manual. Appendix B functionally lists the requests. Appendix C is a glossary of Emacs terms.



File No.: 1L12 CP31-00

One other manual referred to in this book is the *Multics Programmers' Manual* (MPM) — *Commands and Active Functions*, Order No. AG92. It is referred to as the MPM Commands.

Multics Emacs was modelled after the EMACS editor at the Massachusetts Institute of Technology (MIT) Artificial Intelligence Lab. EMACS was written, in TECO, by staff members of the MIT AI Lab and the MIT Laboratory for Computer Science, without whose encouragement and support this project would not have been possible.

CONTENTS

## ILLUSTRATIONS

# SECTION 1

# INTRODUCTION

Multics Emacs is an integrated editing, text preparation, and screen management system designed to take advantage of the features of modern display terminals. Text entry and editing on these video screen display terminals are done interactively. You can see the effects of Emacs editing on the screen as you type.

This manual does not describe all of the editing requests and features available on the Emacs text editor. Instead, a subset of the requests has been selected; those requests most generally used were chosen. Once you have learned them, you can easily progress to the remaining requests and concentrate on those of specific interest to you. All the requests are documented within Emacs itself, and Section 9 tells you how to make use of this "self-documentation" feature.

Many examples of how the requests work are included in each section, but you may find it additionally helpful to try each one out, perhaps as you reach the "Summary of Requests" at the end of the sections describing them. In this way, you will quickly "get a feel for" Emacs.

Throughout this manual, Emacs designates the text editing system, and emacs (all lowercase) designates the Multics command invoked to use the system.

## THE TERMINAL

Emacs has been designed especially for use on a video terminal (often called a CRT, for cathode ray tube). The three parts of the terminal that you will be using as you edit are:

- the screen
- the keyboard
- the modem communicating between the terminal and Multics (unless the terminal is hardwired, i.e., connected directly to the Multics system)

Figure 1-1 shows a typical video terminal and Figure 1-2 shows a typical keyboard and the special keys described below.

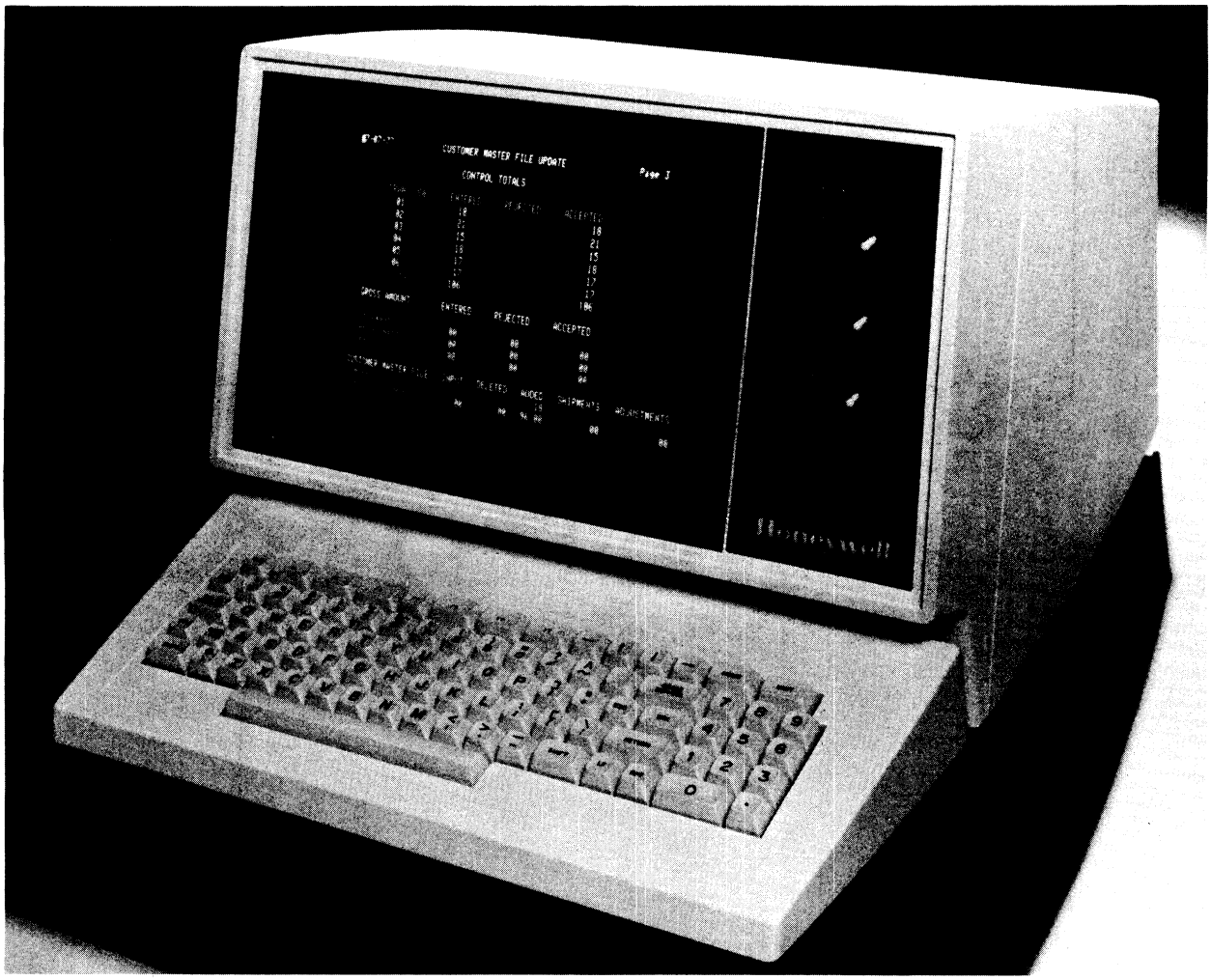



Figure 1-1. A Screen Terminal

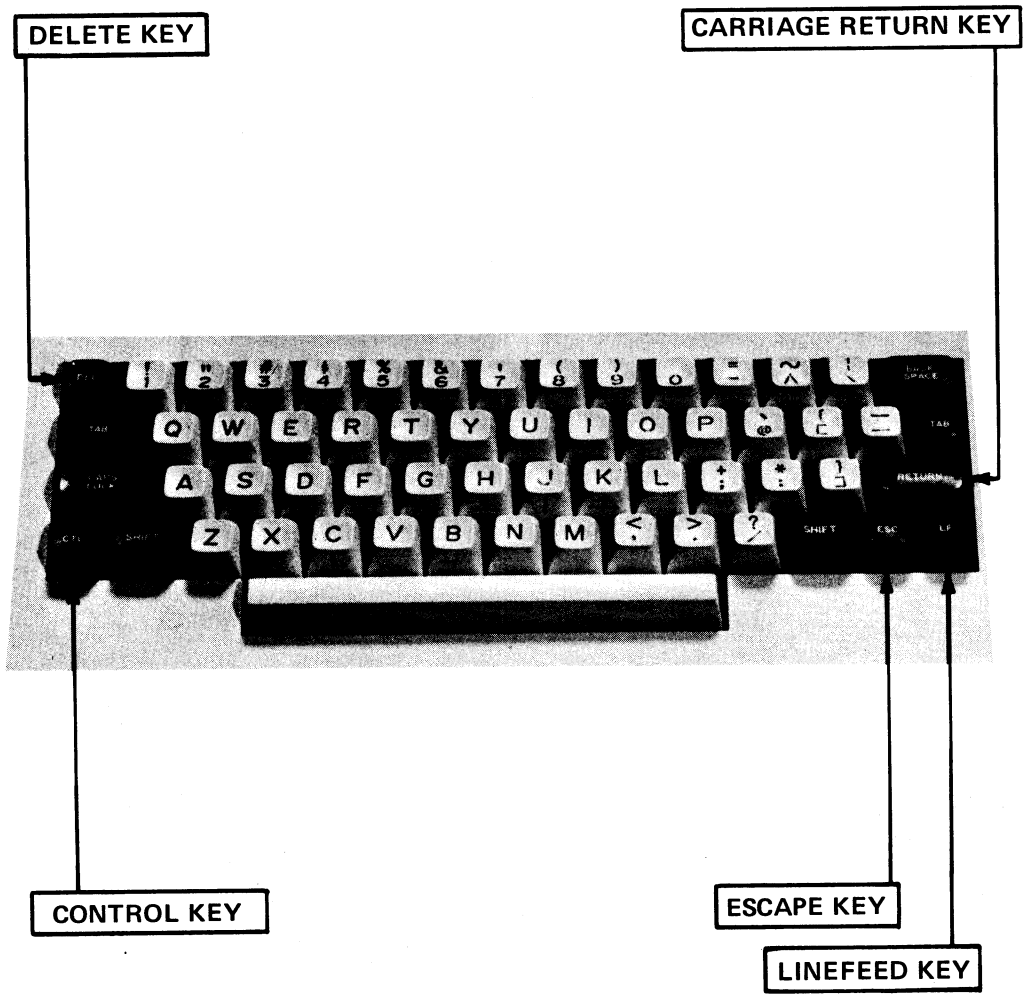



Figure 1-2. A Terminal Keyboard

## The Screen

The screen of your terminal is like a television screen, and displays the information needed to communicate with Multics and Emacs. Messages from the system appear on the screen, and your responses, typed on the keyboard, also appear.

## The Keyboard

Your keyboard resembles the keyboard of a typewriter, with its letters and special characters, but has additional keys. Several of them are important for Emacs usage. They include the following:

- control key
- escape key
- linefeed key
- delete key
- carriage return key

### CONTROL KEY

Terminals vary, but you should be able to locate a key labelled with the letters CTL, CTRL, CONTROL, CNTRL, or something similar. This is the control key. It operates like a shift key in that it must be held down while you hit one or more additional characters. Simply pressing it and releasing it has no effect. For example, if you press the "p" key, you get a lowercase p. If you press the "p" key while holding down the shift key, you get an uppercase P. If you press the "p" key while holding down the control key, you get something called a control P. This control P is a CONTROL CHARACTER. All control characters are interpreted as requests to Emacs. Control characters are used to control Emacs, to manipulate the cursor and the text.

In this manual, the ^ symbol represents the control key; alphabetic characters following the ^ symbol are represented as uppercase, even though, for them, a control character is the same whether the shift key is held down or not (and generally you do not hold it down).

ESCAPE KEY

The escape key is commonly labelled with the letters ESC, ESCAPE, ALT, or ALTMODE. On some terminals, you may have to hold down the shift key to get an escape. Unlike the control key, which is held down while another character is typed, the escape key is typed sequentially, i.e., before or after another character. You use the escape key for some Emacs requests. Always be sure to release the ESC key quickly, to avoid getting two (or more) escapes in a row. Some terminals may not have an escape key; use ^[ instead.

In this manual, the letters ESC represent the escape key.

LINEFEED KEY

Your keyboard should have a linefeed key labelled with the letters LINEFEED, LF, or NEW LINE. The linefeed key is sometimes used in Emacs, and you should turn off auto-linefeed on your terminal if it has it.

DELETE KEY

The delete key is generally labelled with the letters DEL or RUBOUT. As its name suggests, its use is to rub out, or erase, the previously typed character(s). Emacs self-documentation descriptions represent the delete key as the character sequence \177.

CARRIAGE RETURN KEY

On Multics, a carriage return returns you to the left margin and inserts a newline character in your input. This key is often labelled either RETURN or CR.

## The Modem

Your terminal must be connected to the Multics system in some manner for the emacs command, or any other Multics command, to work. Unless the terminal is directly connected (hardwired), the modem or an acoustic coupler provides an interface to the communications link between your terminal and Multics. Many modems are equipped with a telephone receiver and dial. For this type of modem, you dial a specific number to begin the logging in procedure and make the connection to Multics. However, a wide variety of modems exist; if you do not know how to establish the connection between your terminal and the Multics system, you should ask a technically qualified person at your site to help you log in.

## Technical Requirements

Two technical requirements that your terminal must meet are that it be an ASCII terminal, and that it be capable of running in full duplex mode, with no local display of keyboard input (i.e., controllable local echo). Your terminal and/or your modem may have a switch that can be positioned to half or full duplex mode; you should set these switches to full duplex. If your terminal does not have controllable local echo, you should log in in full duplex and echoplex modes (use the echo preaccess comand before issuing the login command). Generally, your site will have arranged for appropriate terminal modes to be set automatically when you log in. If you find characters are printed out twice, setting the modes and/or switches should correct the problem. Again, if you have a question about either of these requirements, ask your site's support staff for help.

If your terminal has an auto-linefeed key or switch, be sure it is off, and use lfecho mode to achieve its effect. Failure to do this results in certain displays vanishing from the screen prematurely.

## LOGGING IN

The first thing you want to do is establish a connection with the computer. This is called logging in. To log in, you must be registered on the system, as a member of a certain project. You are given a unique USER_ID (user identification) that consists of a PERSON_ID (name) and PROJECT_ID (project name). For example, Mary Smith, working in the sales department, may be given the following User_id:

```
Smith.Sales
```

This User_id belongs to Mary alone; no one else can use it. Mary also has a password, which along with her User_id allows her to use the system.

The procedure for logging in is explained in depth in the New Users' Intro--Part I. Briefly, however, to log in you turn power on for the terminal, dial the appropriate telephone number, and when you hear a beep signal, either press a button or place the telephone receiver in the modem and wait. (This method is employed unless your terminal is directly connected to Multics, in which case you do not need to dial a phone number.) When a connection has been established, a header of the following type is displayed by Multics on the terminal:

```
Multics MR8.0: Honeywell LISD Phoenix, System M
Load = 7 out of 95.0 units: users = 7, 02/29/80 1404.3 ...
```

At this point, type the login command and your Person_id, separated by a blank, and then a carriage return (some sites additionally require you to type your Project_id, separated by a space from the Person_id). For example:

```
login Smith
Password:
```

or:

```
login Smith Sales
Password:
```

Multics then requests your password (the second line, above). The display of your password, which you type in after the system-generated Password: line, is suppressed by the system. If you make an error while logging in, the system informs you of it and asks you to try again; you must start again with login:

```
Login incorrect.
Please try again or type "help" for instructions.
login Smith
Password:
```

After you have successfully typed your password, the system responds with information regarding your last login.

```
Smith Sales logged in 06/07/80  0937.5 mst Tue from ...
Last login 06/06/80  1359.8 mst Mon from terminal...
```

The last line of system-generated text in the log-in sequence is the ready message. This message is printed to indicate that Multics is at command level and ready to receive the next command. The ready message consists of the letter r followed by the time of day and two numbers that reflect system resource usage. For more information about the ready message, refer to the ready command in the MPM Commands:

```
r 12:22 3.229 1799
```

The complete log-in sequence for Mary Smith is:

```
login Smith
Password:

Smith Sales logged in 06/07/80  0937.5 mst Tue from ...
Last login 06/06/80  1359.8 mst Mon from terminal...
r 12:22 3.229 1799
```

## INVOKING THE emacs COMMAND

You have logged in and received the ready message indicating that you are at command level. To enter Emacs, type the emacs command on your keyboard, followed by a carriage return (Multics command lines are always terminated by a carriage return):

```
emacs
```

Depending upon the facilities available at your site, Emacs may or may not ask you:

```
What type terminal do you have?
```

It asks this once per session, if it asks at all, and you respond by typing in the name of the type of terminal that you are using. If you type an unacceptable name, Emacs displays the names of all terminals it supports. Among these names, you should be able to find the acceptable form of the name of your terminal. Type it in, followed by a carriage return.

If you cannot find your terminal listed, type:

```
quit
```

followed by a carriage return, to return to command level so that you can log out. You can try again on a different terminal type, or seek assistance.

## The Initial Display

Once Emacs has recognized your terminal type, either automatically or by querying you as described above, it takes several seconds to get started. When it has started up, it clears the screen, and displays the line:

```
Emacs (Fundamental) - main
```

at the lower left of the screen. This line is called the MODE LINE. It tells you several things, the most important of which is that you are talking to Multics Emacs, rather than to the command processor or to another editor. The name of the MAJOR MODE you are in is parenthesized, and here it is Fundamental major mode. Emacs has several modes best suited for different tasks, such as preparing text or programs in the various programming languages. Major modes each have a distinct set of KEY BINDINGS (typed key sequences that specify particular request instructions to Emacs). MINOR MODES provide "fine tuning" to modify the way Emacs works, but do not have a special set of key bindings. The names of minor modes, if you are using any, appear right after the major mode name in the mode line, enclosed in angle brackets (<>). Fundamental mode is the simplest major mode, and the only one described in this manual.

You can edit several things at once with Emacs. Each separate thing being edited is edited in a separate work space called a BUFFER. You actually edit in only one buffer at a time, but you can move from buffer to buffer to work on the contents of each in turn. The buffers are named so that you can differentiate between them. The BUFFER NAME of the buffer you start out in is "main". This name is the last item displayed in the mode line above.

The area between the top of the screen and the mode line is where text appears, and where you deal with the text. This area of the screen is called the WINDOW. The window always displays about twenty consecutive lines (or however many fit on your particular terminal) of the document you are editing. Of course, if your document is shorter than that, part of the window is empty. The MINIBUFFER is the two-line area below the mode line at the bottom of your screen. Emacs uses this space for messages and questions, without interfering with the text displayed in the window. Initially, the minibuffer is empty.

At this point, the only sign of life in your window is a blinking object in the upper left corner. This is the CURSOR. It may be a blinking underline, or a blinking or solid box, depending on your terminal. The cursor is the most important object in Emacs. It is always on some position on the screen, and all "action" occurs at the cursor. All the text you enter is entered at the cursor (and the cursor moves), and any text you delete is deleted at the cursor.

## SUMMARY OF TERMS

When the emacs command is invoked, the first screen displayed is pretty simple, since it is practically empty. You should, however, be familiar with the following terms, and be able to relate them to what appears on the screen.

- control character
- mode line
- major mode
- key bindings
- minor mode
- buffer
- buffer name
- window
- minibuffer
- cursor

# SECTION 2

# ENTRY AND EDITING OF TEXT

## TYPING IN TEXT

Entering text in Emacs is a simple process. After invoking the Emacs command, you begin with an empty buffer named "main". This buffer is your work space; you enter text simply by typing. Whatever you type appears on the screen as you type it, beginning at the top left of your screen at the cursor. As each character is entered, the cursor moves to the right, marking where subsequent characters will appear. When you type the carriage return key for a Multics newline, the cursor moves down a line and to the left edge of the screen, ready for a new line of typing. That is all there is to text entry; characters enter at the cursor as you type. The contents of your buffer are exactly as they appear on your screen.

After the first character is typed into your buffer, an asterisk (*) appears at the bottom of the screen, below the mode line. This indicates that the buffer has been MODIFIED, or changed in some way. When you are just starting out with an empty buffer, this modification, of course, is the change from a buffer with no characters in it, to one with characters in it. The asterisk remains until a copy of the contents of the buffer, which is only a temporary work space, is written out to a file, a segment in the Multics storage hierarchy. The asterisk reappears whenever new modifications occur that have not been written out. Basically, the asterisk lets you know that changes have been made in the buffer and they will not be saved in a permanent file until you write them out. This is discussed more in Section 5.

## EDITING TEXT

Editing is done in Emacs via editing REQUESTS. Each request is a set of programmed instructions interpreted by the editor. You issue, or INVOKE, a request by typing certain keys or key sequences. Each request (key or key sequence) is associated with (i.e., bound to) a COMMAND-NAME, which tells Emacs what set of instructions to follow when that request is typed. Command-names for the requests are hyphenated, abbreviated (sometimes) names intended to suggest the actions of the requests to you, as well as specifying the appropriate instructions to Emacs.

You need an editor to type in new text or programs, correct errors in new or existing text or programs, and make additions, deletions, or changes to text or programs. The cursor's position determines where all of this activity takes place. The particular requests you issue determine what the activity is.

### Self-Inserting Characters

When you type in new text, you are actually issuing Emacs requests. Printing characters (other than #, @, and \, whose special meanings are explained later) are called SELF-INSERTING, because when you type one, it inserts itself into the text in your buffer. So typing the letter "d", for example, tells Emacs:

```
What:   to insert a "d"
Where:  at the cursor
```

Using this type of request is easy.

### Cursor Moving Requests

To make changes in text, you must be able to control the position of the cursor, since that is where action occurs. Many of the Emacs requests, therefore, serve to move the cursor from place to place. For example, when you discover a typing error in the line above your current position in the text, you must move the cursor to that spot before using any requests to correct the error. This manual describes requests that move the cursor:

- forward or backward a character
- forward or backward a word
- to the beginning or end of a line
- to the next or previous line

- to the beginning or end of a buffer
- to the next or previous screen

Positioning the cursor is half the editing battle.

## Deleting Requests

Once the cursor is positioned at the point where you want to make a change, you need requests that actually make the correction. If you are simply adding something, you just type in the addition. Often, however, you first have to get rid of incorrect text. Emacs provides many requests for making deletions. In addition, a facility called the KILL RING saves deleted text so that you can change your mind and retrieve something deleted (killed) in error. The kill ring is useful for moving portions of text, too; you can delete something from point A and reinsert is at point B.

This manual describes requests that delete:

- characters
- words
- lines

In addition, you can delete (and retrieve) a REGION. A region is any extent of text between the cursor and an arbitrary point chosen by you. This arbitrary point is called the MARK; you SET THE MARK by issuing yet another type of Emacs request.

## Other Requests

Emacs has many additional requests that perform other tasks besides moving the cursor and inserting or deleting characters. They help you edit efficiently, format your work, read and write your files from/to the storage system, and of course, return to Multics command level from the Emacs editor. Some requests are provided just to tell you what other requests are available, and how they work. This manual does not cover all of the Emacs requests; however, once you are comfortable with those presented here, you can use these "help" requests to experiment with and learn as many of the remaining requests as you choose. Depending on the nature of your work, some will prove to be invaluable and frequently used editing tools for you, while others may be used only occasionally.

SUMMARY OF TERMS

A few new terms have been introduced that you should remember, since they will be mentioned frequently.

- modified buffer
- request
- command-name
- kill ring
- region
- mark

SECTION 3

# REQUESTS FOR MOVING THE CURSOR

When you log in and invoke the emacs command, you can see the simplest cursor moving requests in action if you type in some sample text. Suppose you type the following:

```
Congratulations  You've just joined
the team of people gushing to learn Emacs
This is sample text created with
the emacs text editor._
```

Though you can, no doubt, supply your own mistakes, a few have already been provided in this example.

Before this text was typed in, the cursor appeared in the upper left corner of the screen. Carriage returns were used to start each new line. As each letter or space (self-inserting characters) was typed, the cursor moved one position to the right as the new letter or space appeared on the screen. The underscore above indicates the cursor's position after the typist enters the final period.

## MOVING THE CURSOR UP OR DOWN A LINE

To change team to stream in the second line, you must first move up to that line from the last line. (In the examples demonstrating the requests for cursor movement, the cursor's position is represented by an underscore (_). In the examples where an underscore is an integral part of the sample text, an underscore representing the cursor appears a line lower than the text.)

## ^P AND ^N

The request for moving up to the previous line is named prev-line-command. You invoke it by first pressing the control key, and holding it down while typing a p, and then releasing both keys. This key sequence (request) is called control p, and is represented as ^P. After typing one ^P, the screen looks like this:

```
Congratulations  You've just joined
the team of people gushing to learn Emacs
This is sample text created with
the emacs text editor.
```

After another ^P, the cursor appears in the second line:

```
the team of people gushing to learn Emacs
```

Note that the cursor stays in the same column as it moves up for each ^P. With this request, the cursor always attempts to stay in the same vertical position. However, if the cursor is moving to a short line and is already beyond the last character of the shorter line, it moves to the position right after the last character on the shorter line, rather than to that directly above its previous place.

When at the first line of your buffer, a ^P simply rings a bell (or beeps) on your terminal, since no previous line exists.

To move down a line, use the ^N request, named next-line-command. This works the same as ^P, except it moves the cursor down a line at a time, and beeps the terminal if issued on the last line of your buffer. Again, you issue this request by holding the control key down while you type an n.

## MOVING FORWARD OR BACKWARD A CHARACTER

While ^P and ^N move the cursor up or down to the line to be corrected, you still must get to the proper place within the line for the correction. One way to move within the line is by moving forward or backward a character at a time.

## ^B AND ^F

The request for moving backward (to the left) a character is ^B, backward-char. The following results when you type a ^B in the example above:

```
the team of people gushing to learn Emacs
```

The cursor moves from the h to the s. Typing three more ^Bs puts the cursor under the space:

```
the team of people gushing to learn Emacs
```

Spaces between words are treated like any other character; tabs and the newline characters at the ends of lines, however, count as only one character. Thus, if the cursor were here:

```
Congratulations  You've just joined
the team of people gushing to learn Emacs
```

it would appear here after the next ^B:

```
Congratulations  You've just joined
the team of people gushing to learn Emacs
```

If the cursor is at the beginning of the buffer (under the C), a ^B beeps the terminal.

The ^F request, forward-char, moves the cursor forward (to the right) one character. Again, tabs and newlines count as one character, and ^F beeps the terminal if at the end of the buffer.

## THE POINT

As mentioned earlier, all action occurs at the cursor, so you move it to the t in team by means of the ^P, ^N, ^B, and ^F requests. To change this to stream, you want to add an s before and an r after the letter at the cursor. In Emacs, when the cursor is properly positioned, you add letters just by typing them in. For which letter is the cursor properly positioned? The cursor's width can confuse you. The cursor's left edge is actually what you want to be aware of. It can be described as being between two characters, the character at the cursor and the character preceding that one (the t of team and the space separating team from the). This left-edge position between characters is called the POINT. Whenever the cursor's position seems ambiguous to you in relation to an editing maneuver, the point's position should resolve the ambiguity.

In the example given, you would type an s, since the character enters at the point:

```
the steam of people gushing to learn Emacs
```

The s is inserted, and the cursor remains under the t as the rest of the line shifts one to the right. Now, typing a ^F and then an r converts steam to stream as desired.

## MOVING TO THE BEGINNING OR END OF A LINE

The line with the cursor in it is called the CURRENT line. You can get to either end of it by means of the next two requests.

### ^A AND ^E

The ^A request, go-to-beginning-of-line, moves the cursor to the beginning of the current line, i.e., moves the point to just before the first character. The ^E request, go-to-end-of-line, moves the cursor to the end of the current line (i.e., after the last character and before the newline). On an empty or blank line, this is the same as the beginning of that line. In the "stream" line, above, suppose you want to add a period at the end. Simply type a ^E:

```
the stream of people gushing to learn Emacs_
```

Then you would simply type in the period:

```
the stream of people gushing to learn Emacs._
```

Typing a ^A while the cursor is anywhere in this line would put it under the t of the.

## WORDS

A word in Emacs is an unbroken string of upper and lowercase alphabetics (a-z and A-Z), numbers, underscores, and backspaces. Lower and uppercase letters can be mixed in any way. For example, "new_payroll", "zeBrA", and "begin" are each one word; "delete-char" and "segname$entry" are each two words.

## MOVING FORWARD OR BACKWARD A WORD

The next two requests are examples of Emacs requests in which the escape key is part of the key sequence. The letters ESC here represent the escape key; when followed by a space and a character (e.g., ESC F), you press and release the escape key and then type the character, but not the space. Alphabetic characters are given in capitals, but you can type either an upper or lowercase letter (as for ^N, ^P, ^B, etc.). Thus, for ESC F, you would type the escape key, release it and type an f or F.

### ESC B AND ESC F

The ESC B request, backward-word, moves the point backward one word.

- If the cursor is currently on some character of a word other than the first character, it moves to the first character of that word (so the point is to the left of the first character).

- If the cursor is on a character between two words (even if they are separated by many blank lines, punctuation, etc.), or on the first character of a word (so that the point is between the two words), it moves to the first character of the preceding word.

Thus:

```
new_payroll,   segname_$entry
                        -
```

after one ECS B becomes:

```
new_payroll,   segname$entry
                      -
```

A word, remember, is an unbroken string of alphabetics, numbers, underscores, and backspaces, so the dollar sign is the first break backward, and the point is left before the word entry.

A second ESC B leaves you here:

```
new_payroll,   segname$entry
               -
```

and a third, here:

```
new_payroll,   segname$entry
-
```

Note that the intervening white space and punctuation are skipped over, and that the underscore in new_payroll is part of that word.

The ESC F request, forward-word, moves the point forward over one word, as you might expect.

- If the cursor is currently on a character that is part of some word, it moves to the first character after that word.
- If the cursor is currently on a character between two words, it moves tothe first character after the second of those two words.

With the cursor as it is in the last example above, an ESC F has this effect:

```
new_payroll,   segname$entry
           -
```

Another ESC F:

```
new_payroll,   segname$entry
                       -
```

One more:

```
new_payroll,   segname$entry
                            _
```

## MOVING TO THE BEGINNING OR END OF THE BUFFER

When editing, you often need to get to the beginning or end of the text in your buffer. Rather than moving through it line by line, you can issue either of the following requests.

### ESC < AND ESC >

The ESC < request, go-to-beginning-of-buffer, puts the point before the first character in the buffer at the top of the document being edited.

The ESC > request, go-to-end-of-buffer, moves the point to the end of the buffer, before the newline on the last line of the document.

## MOVING THROUGH A BUFFER SCREEN BY SCREEN

Since your screen window only displays about twenty lines of text, you need a quick way to view previous or succeeding screens in longer documents. When the cursor is moved about in a buffer containing more than one window of text, Emacs ensures that the current line stays in view. For example, when the cursor is on the last line showing in the window, a ^N causes the screen to clear and be refilled with that next line occupying the middle of the screen, and the surrounding text displayed appropriately. So, if you are viewing lines 51-71, say, and the cursor is on line 71, a ^N moves the cursor to line 72 and displays lines 62-82. The current line (72) shows up in the middle of the window. Emacs always displays the current line appropriately without you worrying about it. However, if you must read through a long document, or locate a section to be edited, a convenient way to view succeeding screens is required.

### ^V AND ESC V

To view the next screen, issue the ^V request, next-screen. The window fills with new text and the cursor moves to the upper left corner, a new position in the buffer. If you type a ^P after a ^V, Emacs chooses a different portion of the buffer to display, centering the line of interest. The cursor always starts out in the upper left corner after a ^V, however. In addition, the first line on the new screen after a ^V is always the same as the last line on the old screen. This helps orient you as you "page" through the text.

To page backward through the text, use ESC V, prev-screen. The previous screen is displayed, and the cursor moved again to the upper left corner. With ESC V, the first line of the old screen is displayed as the last line of the new screen. These two requests provide a convenient way to read through, or rapidly reposition the cursor in, a buffer.

## Moving through a Buffer by Locating Character Strings

You can also move the cursor directly to a specified character string contained in your text. You specify the character string, and Emacs searches for it and moves the cursor there. This is the most direct way to move to a specific point in your buffer, but you must, of course, know rather precisely what you are looking for. The requests for searching for a character string are described in Section 7.

## NUMERIC ARGUMENTS

Requests for which it is useful to specify "how many times" to do them generally accept NUMERIC ARGUMENTS. Several of the requests covered so far fall into this category.

You give a numeric argument by pressing the escape key, releasing it, typing the number you wish, and then typing the request. For example, to move five characters forward or go down four next lines, you would give the ^F or ^N request numeric arguments of 5 and 4, respectively:

```
ESC 5 ^F
ESC 4 ^N
```

In a buffer containing a large document, you could go from line 1 to line 250, for example, by typing:

```
ESC 249 ^N
```

The screen would fill with lines 240 to 260 (approximately), with line 250 and the cursor in the middle of the screen.

## Requests Accepting Numeric Arguments

Of the previous requests, the following accept numeric arguments. Hereafter, as each new request is introduced, its action with a numeric argument, if it accepts them, will be described. Using numeric arguments with requests often greatly speeds your work.

- ^P, prev-line-command, moves up the specified number of previous lines; e.g., ESC 5 ^P moves up 5 lines.
- ^N, next-line-command, moves down the specified number of next lines; e.g., ESC 16 ^N moves down 16 lines.
- ^B, backward-char, moves backward the specified number of characters; e.g., ESC 8 ^B moves backward 8 characters.
- ^F, forward-char, moves forward the specified number of characters; e.g., ESC 3 ^F moves forward 3 characters.
- ESC B, backward-word, moves backward the specified number of words; e.g., ESC 6 ESC B moves back over 6 words.
- ESC F, forward-word, moves forward the specified number of words; e.g., ESC 4 ESC F moves forward 4 words.
- ^V, next-screen, pages forward the specified number of windows and displays it; e.g., ESC 2 ^V moves forward 2 windows.
- ESC V, prev-screen, pages backward the specified number of windows and displays it; e.g., ESC 2 ESC V moves backward 2 windows.
- Self-inserting characters are added to the text the specified number of times; e.g., ESC 10 * inserts 10 asterisks.

### ^X=

The ^X= request, linecounter, is described here because it is often useful to know what line you are on, and how many lines the buffer contains, when choosing a numeric argument. To issue it, you must depress the control key and type an x while doing so, then release both keys and type an equals sign (=). Your terminal may require that you use the shift to get an equals sign. Emacs prints the buffer length, current line number, and the cursor's dprint column position in the MINIBUFFER. The minibuffer is the two-line area below the mode line at the bottom of your screen, and something like this appears in it after a ^X=:

```
138 lines, current = 45, column = 13
```

## ABORTING A REQUEST OR PROMPT RESPONSE

Occasionally, in the act of invoking an Emacs request, you may change your mind about doing so. Requests do not have any effect until you complete the keystroke sequence, so if you have typed the control key only, nothing happens and no harm is done. If you have typed the escape key, however, you may accidentally issue an unwanted request when you resume typing. Or, if the request happens to be one that prompts you for a response, and you have completed the keystroke sequence, how do you escape from the minibuffer without responding?

^G

The ^G request, command-quit, returns the cursor to the current point in the buffer from the minibuffer prompt, if any, and beeps the terminal. The prompting request is aborted. If you have typed the escape key, ^G signals Emacs to ignore it, and beeps the terminal; again, you have aborted any possible accidental request. You can invoke ^G at any time; since it always beeps the terminal, it is useful not only for aborting requests, but also for signalling you that Emacs has finished with previous requests. When you hear the beep, you know Emacs has executed the ^G.

## EXITING FROM THE EDITOR

If you wish, at this point, to log in and practice, you would probably like to know how to get back out of the editor once you have gotten in by invoking the emacs command.

^X^C

To return to Multics command level from Emacs, type the ^X^C request, quit-the-editor. The control key must be depressed while you type the x and the c keys. Since you will be typing text, but not writing it out to storage, you will get this message at the top of your screen after typing ^X^C:

```
Modified Buffers:
> * main
-----------------
```

This message means that the current buffer, indicated by the pointer (>), is modified (*), and its buffer name is "main". At the same time, Emacs writes a message and question, called a PROMPT, in the minibuffer (the prompt is called that because Emacs is prompting you for a response):

```
Modified buffers exist.  Quit?_
```

The buffer becomes "modified" as soon as you make any change to it, and stays modified until you write it out (described in Section 5). This prompt is Emacs' way of ensuring that your modifications are not lost accidentally; you get no prompt after ^X^C if your file has been written out since the last modifications.

Answer this prompt by typing yes followed by a carriage return to terminate the prompt. Angle brackets (<>) appear in the minibuffer at the end of your response when you type the carriage return. Emacs clears the screen and returns you to command level, indicated by the appearance of the ready message at the top of the screen.

If you mistype the yes, Emacs lets you know that your response was inappropriate, by replacing the Modified Buffers line across the top of your screen with:

```
Please answer "yes" or "no"
```

A no answer, of course, simply returns the cursor to its former position in the window, ready for more editing.

The figure below illustrates the commands and requests for logging in to Multics, entering and exiting Emacs, and logging out.

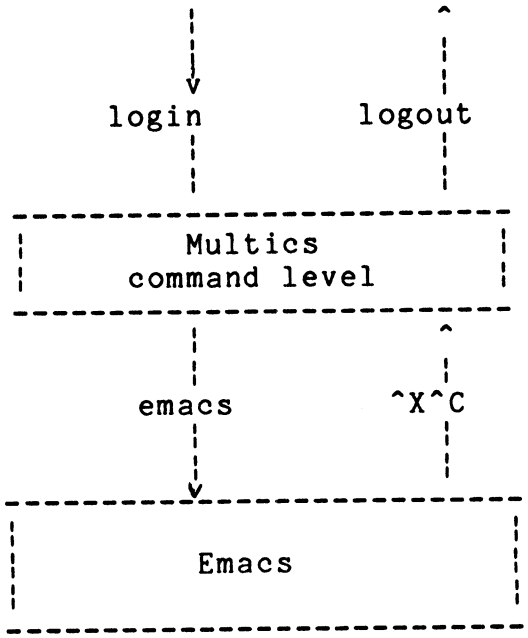



Figure 3-1. Editor Entry and Exit

## Logging Out

Once returned to Multics command level, you invoke the logout command to break the connection between your terminal and Multics. After you have typed it, the system responds by displaying your identification, the date and time of the logout, and the total CPU time and memory units used.

```
logout
Smith Sales logged out 02/29/80  1249.4 mst Fri
CPU usage 17 sec, memory usage 103.1 units.
hangup
```

The word hangup is displayed by Multics to remind you to hang up the telephone and to indicate that the connection has been purposely broken.

## SUMMARY OF TERMS

The new terms introduced in this section are listed below, followed by the requests and their command-names.

- point
- current line
- numeric argument
- prompt

SUMMARY OF REQUESTS

| | |
|---|---|
| ^P | prev-line-command |
| ^N | next-line-command |
| ^B | backward-char |
| ^F | forward-char |
| ^A | go-to-beginning-of-line |
| ^E | go-to-end-of-line |
| ESC B | backward-word |
| ESC F | forward-word |
| ESC < | go-to-beginning-of-buffer |
| ESC > | go-to-end-of-buffer |
| ^V | next-screen |
| ESC V | prev-screen |
| ^X= | linecounter |
| ^G | command-quit |
| ^X^C | quit-the-editor |

# SECTION 4

# DELETIONS

Making corrections or changes in text involves more than just adding characters; you often must DELETE one or more characters. This section describes how to delete single characters, words, and entire regions, and also how to retrieve deleted words or regions and reinsert them, possibly at a different place in your text.

## DELETING A CHARACTER

Quite often, you realize you have made a typing error a split second after you make it. The next requests take advantage of this realization, particularly if your mind works faster than your fingers.

### # AND THE DELETE KEY

The rubout-char request is invoked by typing the # character (pound or number sign) on your terminal. Alternatively, you can type the delete key for rubout-char. Either request deletes the previous character, i.e., the character to the left of the point. When you type a # or the delete key, the character before the cursor disappears, and the cursor, and any following text on the line, moves over one position to the left, closing up the hole made by the deleted character.

So, if you have the following:

```
Congratulations   You've just joined
the stream of people g_
```

typing a # or the delete key while the cursor is right after the g gives you:

```
Congratulations   You've just joined
the stream of people _
```

You can delete backward as many characters or spaces as you want by typing a # to delete them one at a time.

If you catch the error later, either on the same line or beyond it, you must first position the cursor, and then issue the # request. So:

```
the stream of people gush_
```

could be corrected with three ^Bs and then a #:

```
the stream of people ush
```

Then simply type an r followed by an ESC F to get you back to the point where you can continue typing:

```
the stream of people rush_
```

^D

The ^D request, delete-char, deletes the character to the right of the point, i.e., the character at the cursor. Because you are deleting the character at the cursor instead of the character just before the cursor, this request is somewhat easier to conceptualize than rubout-char (#). However, depending on where the error is, use whichever request requires the least effort in repositioning the cursor.

For example, in the last three samples just above, to get from the first of them to the third, you typed three ^Bs, a #, an r, and an ESC F. Using the delete char request, you could type one ESC B, a ^D, an r, and an ESC F. Where three ^Bs were required, one ESC B suffices to position the cursor at the g:

```
the stream of people gush
```

The ^D then removes it and closes up the space:

```
the stream of people ush
```

and the rest of the sequence is the same as for the correction made with rubout-char. With a numeric argument, ^D deletes forward the specified number of characters.

In summary, the rubout-char request, issued by typing either the # character or the delete key

- deletes the character before the cursor, and is most convenient for deleting a character you have just typed.

The delete-char request, issued by typing a ^D

- deletes the character at the cursor.

## DELETING LINES

### @

Some lines are lost causes, and the merciful thing is to kill them outright and start over. To erase everything typed so far on the current line, use the @ request, kill-to-beginning-of-line. You issue it by typing the @ (commercial at sign) character. If you have something like this:

```
To erris hynab, to firgiivc d_
```

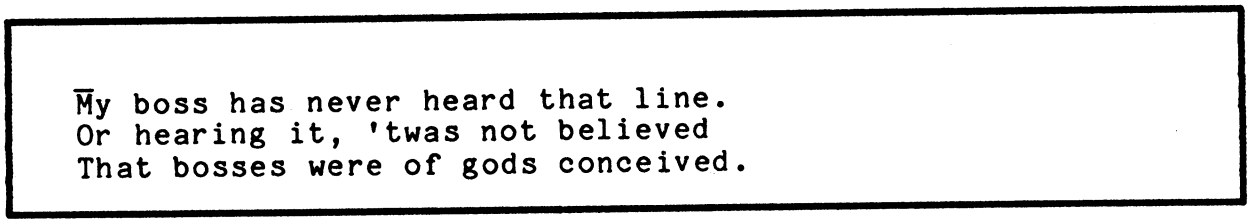
you can type an @ to delete everything to the left of the point, and move the cursor (and any remaining text on the line) to the beginning of the line. In the above, the cursor would move to where the "T" had been, and you could simply retype the line correctly. Text killed with the @ request is saved on the kill ring, described later in this section.

### ^K

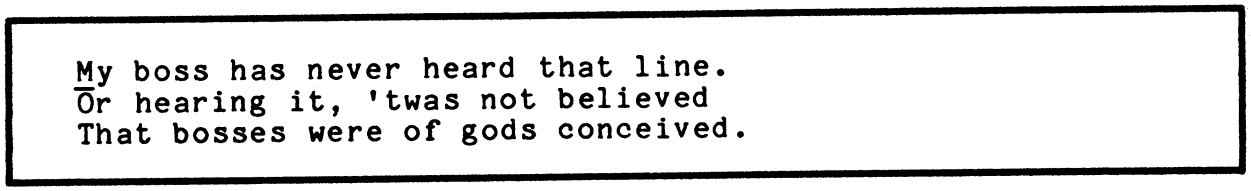
The ^K request, kill-lines, kills forward from the point to the end of the line. If the point is already at the end of a line, ^K deletes the newline, adding the text of the next line onto the end of the current line and moving succeeding lines up one. If the line is empty, the point can be considered as at the end of the line, since the beginning and end are the same, so ^K acts as above, effectively deleting the blank line. Some examples follow:

```
To err is human, to forgive divine
My boss has never heard that line.
Or hearing it, 'twas not believed
That bosses were of gods conceived.
```

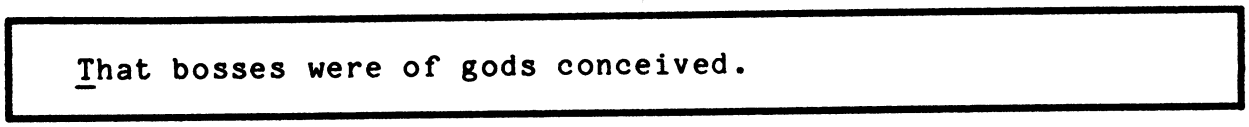
To remove these lines, first type a ^K:




Another ^K removes the now blank top line, and the remaining three move up:




The ^K request accepts a numeric argument, and kills the specified number of entire lines, starting at the current point on the current line. You could get rid of the top two lines (two lines and the two newlines) by supplying an argument of 2, i.e., ESC 2 ^K, to yield:




Whereas ESC 4 ^K kills four lines, four ^Ks would not, since you generally must type two ^Ks to kill one line of text.

You can kill just part of the line (the part to the right of the point):

```
I'd give an arm and a leg, and my eyeteeth,
for a piece of your peanut brittle.
```

After one ^K, this looks like:

```
I'd give an arm and a leg
for a piece of your peanut brittle.
```

After a second ^K:

```
I'd give an arm and a legfor a piece of your peanut brittle.
```

If you then need to reenter text (as in the above example, where deleting the first comma necessarily deleted the space after it), the cursor is conveniently positioned for doing so, or for entering additional text.

```
I'd give an arm and a leg, Peg, for a piece of your <...>.
```

is gotten simply by typing a comma, a space, "Peg,", and another space.

Lines killed with the ^K request are also saved on the kill ring.

## DELETING A WORD

### ESC # AND ESC DELETE KEY

Words can be deleted in much the same way that characters can, i.e., either the word or character to the left or right of the point can be deleted. To delete the last word you typed, the word left of the point, you use ESC # or ESC delete key. Both of these key sequences invoke the rubout-word request, and you type the escape key followed by either the # character or the delete key. For convenience, only ESC # will be mentioned further, but the ESC delete key request is identical. These two requests delete the word to the left of the point, or that part of the word to the left of the point if the cursor is in the middle of a word. In other words, they delete all text between the point and the place where the point would be moved by an ESC B, backward-word.

So, if the cursor is immediately after a word, ESC # deletes only the characters of the word it follows:

| three Frenched_hens | becomes | three _hens |
|---|---|---|

If the cursor is in the middle of a word, ESC # deletes that part of the word to the left of the point:

| one e_bony pony | becomes | one _bony pony |
|---|---|---|

If the cursor is at any other point, ESC # deletes all characters between the point and the preceding word, as well as that preceding word (intervening punctuation and white spaces are deleted, too):

| two turtles, _doves | becomes | two _doves |
|---|---|---|

These two requests accept a numeric argument, deleting backward the specified number of words. Deleted text is saved on the kill ring.

ESC D

The ESC D request, delete-word, deletes the word, or part of a word, to the right of the point, i.e., to the right of, and including the character at, the cursor. It deletes forward from the point to the place where the point would be moved by an ESC F, forward-word.

Thus, if the cursor is on the first character of a word, ESC D deletes the entire word:


If the cursor is in the middle of a word, ESC D removes all the characters from the one at the cursor to the last character of the word:


If the cursor is between words, ESC D removes all white space and punctuation up to the second word, as well as the second word:


The ESC D request accepts a numeric argument, deleting forward the specified number of words. Deleted text is saved on the kill ring.

## DELETING A REGION

A region is the extent of text between the current point and a mark you set to delimit the other boundary of the region. Before you can delete an entire region, you must set the mark to define it.

^@

To set the mark, you position the point by moving the cursor to where you want it, and then issue the ^@ request, set-or-pop-the-mark. If your terminal requires a shift to get the commercial at sign, this may be tricky, since you have to hold both the control and shift keys down while you type an "@" character. Also, some terminals, e.g., those of Digital Equipment Corp., require that you press the control key while typing the space bar to transmit a ^@. After you set the mark, the word "Set" appears in the minibuffer to let you know the mark is set.

Since your purpose is to define a region, you next reposition the cursor to the other "end" of the region, i.e., position the point after the last character to be included in your region.

## Exchanging the Mark and the Point

^X^X

Once the mark is set at one end, and the point positioned at the other, you are ready to delete the region. Before doing so, however, you can verify that the mark is indeed set where you want it. The request for this is ^X^X, exchange-point-and-mark. The cursor moves to where it was when you first set the mark, but the region stays defined because the mark is now set at the former point. Another ^X^X exchanges the mark and point again so that everything is as before. However, the second exchange is unnecessary, although it may be visually reassuring.

Suppose you want to change the part of the text following created below:

```
This is sample text created with
the emacs text editor.
```

The first step would be to set the mark by positioning the point before the w and issuing the ^@ request:

```
This is sample text created with
the emacs text editor.
```

The only thing that happens on the screen after you type the ^@ request is that the word Set appears in the minibuffer.

The next step is to move the cursor to the end of the region to be deleted:

```
This is sample text created with
the emacs text editor._
```

Now you can type ^X^X to verify that the mark was set where you wanted it:

```
This is sample text created with
the emacs text editor.
```

^W

To delete, or "wipe" out, a region, use the ^W request, wipe-region. All the text between the mark and the point disappears from the screen, including any newline characters within the region. One ^W transforms the above to:

```
This is sample text created _
```

Any succeeding lines would move up under this one.

Text deleted with a ^W request is saved on the kill ring.

## RETRIEVING TEXT

The @, ^K, ESC # or ESC delete key, ESC D, and ^W requests each put deleted text on the KILL RING, from which you can retrieve it immediately or even many requests later.

The kill ring has ten "slots" for saving your text. When you issue one of the above requests, the text "killed" goes into the first slot. (The #, delete key, and ^D requests do not make use of the kill ring, since characters deleted singly are easy to retype if you want them back). If you later issue another kill request, the text previously killed moves into the second slot, and the newly killed text goes into the first slot just vacated. Killed text keeps rotating down a slot in this fashion until all ten slots are filled. At the next kill request, the first killed text would be discarded, since no eleventh slot is available.

KILL MERGING is a feature that allows you to save related killed text in the same slot in the kill ring. Kill merging occurs when you type SUCCESSIVE kill requests, i.e., no intervening keystrokes occur between the kill requests. You cannot type in any new text or issue any other type of request. Text killed by such successive requests is merged and saved in one slot.

Thus, an unbroken string of ESC # or ESC delete key, ESC D, ^K, and ^W requests merges all the text killed. Although the @ request saves text on the kill ring, it does <u>not</u> participate in kill merging.

^Y

Kill merging is important because the ^Y request, yank, retrieves text from the kill ring. It is useful for repairing mistakes and for moving lines and regions around. With no numeric argument, ^Y retrieves text from the first slot of the kill ring and inserts it at the cursor. With a numeric argument, ^Y yanks text out of the slot on the kill ring that corresponds to the number given. Thus, if you want, for example, the second latest thing killed (from the second slot), you would type ESC 2 ^Y.

Some examples follow - assume that the mark is already set before the w of with in the top line:

```
This is sample text created with
the emacs text editor._
Requests for moving the cursor
or deleting text
are available, too.
```

You have already seen that a ^W at this point gives you:

```
This is sample test created _
Requests for moving the cursor
or deleting text
are available, too.
```

The cursor is in position, so you can type in a change:

```
This is sample text created for Emacs practice._
Requests for moving the cursor
or deleting text
are available, too.
```

What happens if you position the cursor at the end of the second line and try another ^W? The mark is still set, and remains set until you set it somewhere else with ^@. So, you would get:

```
This is sample text created _
or deleting text
are available, too.
```

You can retrieve the most recent kill by typing a ^Y:

```
This is sample text created for Emacs practice.
Requests for moving the cursor_
or deleting text
are available, too.
```

The cursor is left by ^Y <u>after</u> the text retrieved.

Typing another ^Y would not retrieve the text deleted by the first ^W, since the latest killed text remains in the first slot until another kill moves it to the second slot, whether it has been yanked by ^Y once, never, or many times. Thus, another ^Y here would simply reinsert another copy of what was just yanked at the cursor. This can be useful if you have some text to be repeated, since you can delete it and then yank it back into all the different places you want it. You need ESC 2 ^Y to retrieve from the second slot:

```
This is sample text created for Emacs practice.
Requests for moving the cursorwith
the emacs text editor._
or deleting text
are available, too.
```

To avoid problems like the above, remember that ^Y inserts retrieved text at the cursor (which was at the end of the second line), and leaves the cursor after the retrieved text.

To see how kill merging works, suppose you put the cursor under the w of cursorwith and type an ESC D:

```
This is sample text created for Emacs practice.
Requests for moving the cursor_
the emacs text editor.
or deleting text
are available, too.
```

You then realize the next line (third line) is unwanted, too, so type a ^K:

```
This is sample text created for Emacs practice.
Requests for moving the cursorthe emacs text editor.
or deleting text
are available,too.
```

That killed only the newline, so you type another ^K:

```
This is sample text created for Emacs practice.
Requests for moving the cursor_
or deleting text
are available, too.
```

Perhaps now you decide the next line should go, too, and type two more ^Ks:

```
This is sample text created for Emacs practice.
Requests for moving the cursor_
are available, too.
```

Now, moving the cursor to the end of the buffer (ESC >) and adding a few new lines (carriage returns) gives you a chance to see just what has been merged. A ^Y yanks this text onto the screen, below the remaining lines, at the cursor:

```
with
the emacs text editor.
or deleting text_
```

Since this text was killed by typing only ESC D, ^K, ^K, ^K, and a last ^K, with no other keystrokes between, all the text merged for retrieval with a single ^Y.

In practice, the text you delete with successive deleting requests generally is related, since you will not be yanking fragments into the wrong places (hopefully) as was done above. You may delete several lines with ^Ks, or several words with ESC #s and ESC Ds, and decide you want them back. Typing one ^Y is far less tedious than having to type several to reconstruct your text.

## Popping the Mark

A MARK RING, similar to the kill ring, saves marks set by the ^@ request, set-or-pop-the-mark. It "remembers" the last ten marks set; you can move the cursor to each of these previous marks, in turn, by giving any numeric argument to the ^@ request. This is called popping the mark from the mark ring, and the popped mark is reset as the CURRENT mark, i.e., the mark used by ^X^X or ^W. For example, ESC 1 ^@ repositions the cursor at the position of the second-most recently set mark, and resets it as the current mark. If text was deleted from around that position, the current mark is set at the position closest to its former position. Another ESC 1 ^@ repeats this process (going back to what had been the third most recently set mark). You can move back through the marks, "rotating the ring," by doing this as many times as you wish. You may consider setting marks at strategic places in your text just so you can return to them easily, via ^@ with a numeric argument, for review, further editing, or some other special reason.

## LITERAL CHARACTER ENTRY

Since the # and @ characters are themselves requests, you may be wondering how to get them into your text instead of having them delete part of it.

^Q

To enter these characters literally, precede them with the ^Q request, quote-char. Alternatively, you can precede them with a backslash (\), the escape-char request. Since the \ character is an Emacs request, to enter it literally into your buffer you must precede it with ^Q or a \, too, just as you do for # or @. Thus:

| To insert | Type |
|---|---|
| # | ^Q# or \# |
| @ | ^Q@ or \@ |
| \ | ^Q\ or \\ |

SUMMARY OF TERMS

The terms covered in this section are:

- kill ring
- kill merging
- successive kill requests
- mark ring

SUMMARY OF REQUESTS

| | |
|---|---|
| # | rubout-char (to left of point) |
| delete key | rubout-char (to left of point) |
| ^D | delete-char (to right of point) |
| @ | kill-to beginning-of-line |
| ^K | kill-lines |
| ESC # | rubout-word (to left of point) |
| ESC delete key | rubout-word (to left of point) |
| ESC D | delete-word (to right of point |
| ^@ | set-or-pop-the-mark |
| ^X^X | exchange-point-and-mark |
| ^W | wipe-region |
| ^Y | yank |
| ^Q | quote-char |
| \ | escape-char |

# SECTION 5

# FILES

The purpose of any editor is to provide the means for creating and editing files that you wish to save as segments in the Multics storage system. Thus, you must be able to read files into Emacs from storage, and write them out (or back out) into storage from Emacs.

## WRITING A FILE OUT

When you invoke the emacs command, you automatically start out in the buffer named "main". As soon as you modify this buffer's contents, an asterisk appears below the mode line, indicating that the buffer's contents have changed. To save the new version of the file in the buffer, you must write it out.

### ^X^W

The ^X^W request, write-file, writes the buffer's contents to a storage segment. After typing a ^X^W, you are prompted in the minibuffer for the pathname of the segment to which you want the file written:

```
Write File: _
```

Type the pathname (you can type a relative or an absolute pathname) in response to the prompt, and terminate it with a carriage return. (If you make a mistake while typing in the minibuffer, you can edit it as you would text.) for example, you can type "first.practice" to create a segment named first.practice in your working directory. As Emacs writes this file, the word Writing... appears in the minibuffer:

```
Write File: first.practice<>
Writing...
```

When the word Written replaces Writing..., the file has been successfully written. At the same time, the absolute pathname of your file appears right below the mode line. This is called the PATH LINE, and always tells you what file you are working with. The asterisk disappears, since the copy of the file in the buffer contains no modifications not written out to the stored segment.

The bottom of the screen now looks something like this:

```
Emacs (Fundamental) - main
    >udd>Sales>Smith>first.practice
    Write File: first.practice<>
    Written.
```

When you make subsequent changes, the asterisk reappears in the path line to indicate that the buffer's contents are again modified. In addition, the word Modified replaces Written.

```
Emacs (Fundamental) - main
  * >udd>Sales>Smith>first.practice
    Write File: first practice<>
    Modified.
```

The first time you use ^X^W, you establish the buffer's DEFAULT PATHNAME. When Emacs expects you to supply certain information, and you do not, it uses, by default, the last appropriate information that you did supply. Thus, if you have supplied a pathname already for ^X^W, and wish to write new modifications out to the same segment, you can use ^X^W again and type only a carriage return (a null response) when it prompts you for the pathname. Emacs then uses the default pathname established previously.

^X^S

The ^X^S request, save-same-file, works like ^X^W, except that it never prompts for a pathname. Instead, it always writes the buffer's contents out using the default pathname. A default pathname, of course, must be available, so if you are creating a new file, the first write-out must be done with ^X^W to establish the default; subsequent write-outs can be done with ^X^S. However, the request for reading a file in (discussed below) also sets the default pathname, so using ^X^S to write such a file out is faster than using ^X^W. You never wait for a prompt, and you never have to respond to one (even if only to type a null response). Saving a keystroke here and there may seem trivial, but it is not when you consider that you should write your work out frequently to ensure that it gets saved.

## READING A FILE IN

Every file read in goes into its own buffer, and you can have as many buffers in Emacs as you wish. Suppose you are creating a new document in the "main" buffer, and decide you would like to look at a file prepared and stored previously.

^X^F

The ^X^F request, find-file, reads a file into a buffer. It prompts you for the pathname (relative or absolute) of the file you want:

```
Find File: _
```

You type in the pathname, and terminate the prompt with a carriage return (all Emacs prompts are terminated by CR), for example:

```
Find File: first.practice<>
Reading...
```

The word Reading... appears in the minibuffer while the file is being read; it disappears when the file is read, and the screen fills with the first twenty lines or so of the file. Although the entire file does not appear on the screen, the buffer does contain all of it. The cursor is at the first character of the first line.

The mode line changes because you are no longer in the "main" buffer (although that buffer and its contents still exist). When no buffer exists that already contains the file you ask for, ^X^F reads the file into a new buffer whose name is taken from the first component of the entry portion of the file's pathname (the entry portion is the part of the pathname following the last > sign). In this case, the file is first.practice, so the buffer is named "first". It also sets the default pathname for the new buffer to the file's pathname. Below are the mode line and path line as they appear after reading in first.practice:

```
Emacs (Fundamental) - first
    >udd>Sales>Smith>first.practice
```

You can now read or edit this file. If you make changes to it, you can write it out with ^X^S, since ^X^F set its default pathname. Alternatively, you can write it out to a different pathname, with ^X^W, if you wish to save the old version of it in the original segment and store the new version in a different segment.

If you already have one or more buffers containing the file you are reading in again with ^X^F, Emacs lists these buffers on your screen in a LOCAL DISPLAY. A local display consists of information displayed at the top of your screen that temporarily replaces the text being edited. A line of dashes and asterisks is also displayed beneath the information so that you know this is a local display and your buffer has not been destroyed. You are then prompted for the name of the buffer you wish to use. If the buffer specified in your response is one of those listed, ^X^F switches to it. If a new buffer name is specified, ^X^F reads the file into that buffer. A null response (just a carriage return) switches you to the original buffer named by the first component of the entry portion of the file's pathname (i.e., the buffer entered when you read the file in for the first time).

^X^R

The ^X^R request, read-file, is useful when you wish to undo, quickly, any modifications you have made since the last write request. This request prompts you for a file's pathname and reads that file into the current buffer, replacing whatever was in the buffer previously. It also sets the buffer's default pathname to the pathname of the file read. If a null response is given to the prompt, ^X^R reads the buffer's default file (this is usually what you want) set by a previous ^X^F, ^X^R, or ^X^W.

## Access Restrictions

Sometimes you may encounter a problem when you try to write out a file that you have read in without any trouble. This occurs if you have read (r) access to the file, but do not have write (w) access (access requirements are discussed in the New Users' Intro - I). An error message like this:

```
Incorrect access on entry.
```

appears in the minibuffer in such a case. If you get such a message, you cannot use ^X^S to write the file out; you must use ^X^W and supply a different pathname from the one used to read in the file.

You may get the same message when reading a file in if you do not have read access to it. If this occurs, you must request access from the user whose segment you are attempting to read.

## INSERTING A FILE

^XI

The ^XI request, insert-file, inserts a file into the current buffer at the point. You are prompted for the pathname of the file to be inserted, and the file is read into the buffer without destroying its previous contents; lines following the inserted file are simply moved down below it. The cursor is left after the newline at the end of the inserted file's contents. This request does not change the default pathname for the buffer. For example, if you read in "first.practice" with ^X^F, you can position the cursor wherever you want and issue a ^XI. Responding to the prompt:

```
Insert File: _
```

with second.practice and a carriage return inserts a copy of the contents of "second.practice" at the indicated spot (succeeding lines of "first.practice" are moved down to accommodate the inserted text). The default pathname is still "first.practice." You can insert the same file in your text at many places, or "assemble" many different files into one file, by using ^XI.

## EDITING MULTIPLE BUFFERS

At the beginning of this section, you were in the "main" buffer, creating text. To take a look at the first.practice file, you read it in to the "first" buffer, and were automatically switched to that buffer. You can read in as many more files as necessary, and each goes into its own buffer as you are switched to it. You can also read the same file into several separate buffers; if you issue the ^X^F request, and then type first.practice in response to its prompt again, Emacs lists the buffers containing this file, only the "first" buffer in this case, and prompts for a buffer name. You may decide to use a new buffer for this copy of the file, and type the name you want assigned to it, for example, one.

If you issue the ^X^F request and type first.practice in response to its prompt a third time, Emacs lists both "first" and "one" as buffers containing the file (although they may contain different versions of first.practice if you have edited these buffers). At this point, you can use yet another buffer by typing a different name, or reuse one of these two by typing its name. If you type just a carriage return, Emacs returns you to the buffer "first", filling the window with the contents of that buffer as they were when you last used it.

### Switching Buffers

So, suppose you now have three buffers: main, first, and one. How can you move between them?

^XB

The ^XB request, select-buffer, prompts you for the name of the buffer to which you want to go. To issue this request, you must hold the control key down while typing the X, and release it before typing the b key. You then type the buffer name in response, and Emacs switches to that buffer. The line that was current when you left that buffer is centered on the screen, with the cursor returned to the last current point.

If you give the name of a buffer not yet created, Emacs creates it. You see a blank screen, since the new buffer is empty, with the cursor placed in the upper left corner.

If you type only a carriage return in response to ^XB's prompt, you return to the last buffer you were in before entering the current buffer.

Whenever you switch buffers, the mode line and path line change to reflect the name and path of the buffer switched to.

## Listing Buffers

^X^B

To list the buffers in use in an Emacs editing session, issue the ^X^B request, list-buffers (hold down the control key while typing both the x and b keys). A list of buffers appears as a local display:

```
Listing of current Buffers

>*one                    >udd>Sales>Smith>first.practice
  first                  >udd>Sales>Smith>first.practice
 *main


-- * * * * * * * * * * * * * --
```

The display contains the name of each buffer and the pathname of the file in it, if any. An asterisk (*) before a buffer's name indicates a modified buffer. The greater-than sign (>) indicates the current buffer.

## Restoring the Screen after a Local Display

You remove a local display from your screen by typing the linefeed key. If your terminal has an auto-linefeed switch, it should be "off" to prevent a local display's automatic and premature removal. When the local display disappears, the part of your text that was under it reappears.

## EXECUTING A MULTICS COMMAND FROM WITHIN EMACS

You may have some reason to invoke a Multics command while you are editing in Emacs. You can avoid writing your files out, exiting Emacs with ^X^C, invoking the Multics command, and then reentering Emacs and reading your files in again. Two requests are provided to do this.

## ^X CARRIAGE RETURN AND ^X^E

The ^XCR request, eval-multics-command-line, is issued by typing a ^X and then the carriage return key. It prompts you for a Multics command line, and executes it. It can be useful for executing many Multics commands, for example:

| | |
|---|---|
| set_acl | to give someone access to your files |
| defer_messages | to defer interactive messages |
| rename | to change the name of a file |

When you wish to execute a Multics command that produces terminal output, use the ^X^E request, comout-command. This request executes the Multics command line (also prompted for as above) and displays the command line's output in a buffer named "file_output". After examining the output, you can write it to a file if you want, and switch back to your working buffer with ^XB. Some Multics commands you may wish to execute with ^X^E include:

| | |
|---|---|
| list | to list your segments, perhaps prior to reading one in |
| who | to determine what people are currently logged in |
| print_motd | to see the message of the day |
| dprint | to print a "hardcopy" of a (used with ^X^E because Multics prints a line giving you the status of your dprint request). |

## CLEARING AND REDISPLAYING THE SCREEN

### ^L

The ^L request, redisplay-command, clears the screen and refills it with the same text, except that the current line is centered (unless the current line is already at or near the first line of the buffer). Whenever something invalidates your screen's contents, e.g., an incoming message or random characters from a bad telephone line, ^L restores the screen. With a numeric argument, ^L redisplays the screen with the current line the specified number of lines below the top of the screen, e.g., ESC 1 ^L puts the current line at the top, ESC 6 ^L puts the current line six lines from the top, etc. This is helpful when you want to reposition the lines on the screen.

SUMMARY OF TERMS

The terms introduced in this section include:

- path line
- default pathname
- local display

SUMMARY OF REQUESTS

| | |
|---|---|
| X W | write-file |
| X S | save-same-file |
| X F | find-file |
| X R | read-file |
| XI | insert-file |
| XB | select-buffer |
| X B | list-buffers |
| XCR | eval-multics-command-line |
| X E | comout-command |
| L | redisplay-command |

# SECTION 6

# SPACING AND FORMATTING

This section includes Emacs requests that help you manage the appearance or format of your text.

## FILL MODE

FILL MODE is an Emacs MINOR MODE that allows you to type text into a buffer without ever typing the carriage return key. Text is broken automatically at the end of each line so that is does not extend past the FILL COLUMN, the right margin. Typing a space, tab, or punctuation mark following a word that extends past the fill column signals Emacs to "back up" to the white space preceding that word and end the line there. Thus, you end up with a ragged right margin.

In addition, the FILL PREFIX, if set, is automatically inserted at the beginning of each new line when fill mode is on. The fill prefix determines the left margin, and is empty unless you set it to contain some combination of spaces and characters (see "Margins" below for setting the fill prefix and fill column). The default fill prefix is empty, i.e., the left margin is the left edge of your screen. When not in fill mode, the fill prefix is inserted only when you type a carriage return.

While in fill mode, if you want to end a line before the fill column, type a carriage return at the end of the line. If you want to extend past the fill column, precede each character after the fill column with ^Q. Or, you can turn off fill mode for the line(s), and turn it back on after typing the line(s).

### ESC X fillon and ESC X filloff

The ESC X fillon request turns fill mode on in the current buffer, and the ESC X filloff request turns it off in the current buffer. These two requests are EXTENDED REQUESTS. An extended request is one that is not frequently used during editing, so it is not bound to a key. Instead, you issue the ESC X request, extended-command, by typing the escape key and then the x key. It prompts you for a command-name (and arguments in those cases requiring them), which you type in the minibuffer. Extended requests, are called that because they are issued by typing the ESC X request.

Thus, for example, to turn fill mode on, you type ESC X and are prompted:

```
Command: _
```

You then type the command-name, fillon, and a carriage return:

```
Command: fillon<>
```

All text entered in this buffer after this mode is turned on is "filled" as you type, until you issue the ESC X filloff request to turn this mode off. You must turn fill mode on/off in each buffer where you want to use/stop using it.

## MARGINS

Your right and left margins are generally set automatically to accommodate your terminal's screen size. However, you can change them. The left margin is changed by setting the fill prefix, which is inserted automatically at the beginning of every line by carriage return or fill mode. The fill prefix can contain both spaces and characters, although you generally want only spaces or tabs. The right margin is determined by the fill column.

^X.

The ^X. request, set-fill-prefix, sets the fill prefix in the current buffer. Before typing the ^X., you position the cursor on a line, and whatever combination of spaces and characters fall between the cursor and the screen's left edge becomes the new fill prefix. For example, if you want a fill prefix of five spaces, put the cursor on the sixth character of a line beginning with at least five spaces and issue this request. To reset the fill column to the left screen edge, issue ^X. at the beginning of a line (^A gets you there).

^XF

The ^XF request, set-fill-column, sets the fill column in the current buffer. Before typing the ^XF, you position the cursor in the column that you want as the fill column. Fill mode thereafter uses this column as the right margin. When you set the fill column, its value is displayed in the minibuffer:

```
fill column = 65
```

This request accepts a numeric argument that is the value to be assigned to the fill column. Thus, ESC 65 ^XF sets column 65 as the fill column, and column 64 becomes the last column in which text can be placed when you are in fill mode.

The fill column can be set to a column not on your screen (if you want copies dprinted on wide paper, for example, and wish to save time and paper). If you set the fill column for something like this, your text will necessarily appear on the screen with many continuation lines (\c appearing at the beginning of the lines, with words broken randomly).

## Adjusted Right Margin

Since fill mode gives you a ragged right margin, what do you do to get an adjusted right margin instead (like the one in this manual)? You must format paragraphs individually. Paragraphs can be defined in two ways in Emacs.

ESC X opt paragraph-definition-type

The ESC X opt request is an extended request that sets various options. One of the options is paragraph type; two types are available. The first, the default type, basically defines paragraphs as the text between two blank lines. Type 1 paragraphs always begin at the beginning of a line, and the lines must be:

- preceded by an empty line, or
- the beginning of the buffer, or
- preceded by a Multics runoff or compose control line

(The Multics runoff and compose commands are text formatters; each control line is itself a paragraph.)

If you change the paragraph type, you can change it back to this default type by typing:

```
ESC X opt paragraph-definition-type 1
```

Typing ESC X prompts you in the minibuffer for the command-name, opt in this case. You must also supply any arguments that the extended request takes, and the arguments required here are paragraph-definition-type and 1. You must also type the spaces separating the arguments.

The second type of paragraph begins on any indented line, i.e, any spaces or tabs at the beginning of a line begin a paragraph, and the paragraph ends with the last character on the line preceding the next indented line. So, paragraphs of type 2 begin at lines that are:

- indented, or
- the beginning of the buffer, or
- preceded by a Multics runoff or compose control line

You set the paragraph type to 2 by typing:

```
ESC X opt paragraph-definition-type 2
```

ESC Q

The ESC Q request, runoff-fill-paragraph, fills the current paragraph in the same way that fill mode would. You can use it to format your work as you complete each paragraph; formatting at these longer intervals is slightly faster than fill mode line-by-line formatting.

With any numeric argument, e.g., ESC 1 ESC Q, the paragraph is formatted with an adjusted right margin, with padding where necessary.

With or without a numeric argument, ESC Q uses the fill prefix and fill column for formatting. It also puts the original, unformatted paragraph in the kill ring, so you can retrieve it if you do not like the formatted version.

## INSERTING BLANK LINES

^O

To quickly open up some space so that you can add several lines or paragraphs without waiting while Emacs continually rewrites lines, as it would if you just typed the new text in, use the ^O request, open-space.

This request puts a newline into your buffer ahead of the current point. Text after the cursor point moves down a line and over to the left margin, and pushes succeeding lines down one. The cursor remains where it was when you issued the ^O, so that space opens up below it, and it stays above the inserted blank lines, in position for you to enter the new text.

To insert several blank lines, issue this request with a numeric argument. For example, ESC 4 ^O adds 4 blank lines, as below:

```
Once there were four rabbits,
and they lived in a hollow under the hill.
```

becomes, after ESC 4 ^O:

```
Once there were four rabbits,
-    <blank line>
     <blank line>
     <blank line>
     <blank line>
and they lived in a hollow under the hill.
```

Then you type in the new text:

```
Once there were four rabbits,
Flopsy,
Mopsy,
Cottontail,
and Peter,_
and they lived in a hollow under the hill.
```

Any extra blank lines can then simply be deleted.

## INDENTATION

ESC I

The ESC I request, indent-relative, allows you to prepare outlines, programs, and tables easily. If the current line is not indented, ESC I indents it the same way as the previous non-blank line:

```
Today's agenda:
    Division reports
Sales quotas_
```

becomes

```
Today's agenda:
    Division reports
    Sales quotas
```

As you can see, the cursor is left at the column following the inserted spaces.

If the previous non-blank line and the current line are not indented, ESC I indents the current line to the column occupied by the first letter of the second word of the previous line:

```
May Sales Quotas:
Atlantic Div.
```

becomes

```
May Sales Quotas:
    Atlantic Div.
```

If the current line is already indented (possibly by an ESC I), ESC I reindents it to the column occupied by the first letter of the next word to the right in the previous non-blank line:

```
Quotas:
      Atl. Div.
      $120.5K_
```

becomes

```
Quotas:
      Atl. Div.
           $120.5K
```

With any numeric argument, e.g., ESC 1 ESC I, this request indents the current line like the last previous line having less indentation. Consider the next two examples:

```
Quotas:
      Atl. Div.
           $120.5K
Pac. Div._
```

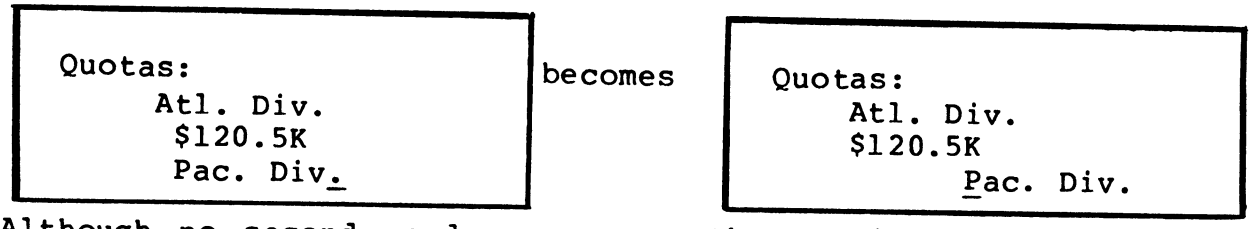
after ESC I becomes

```
Quotas:
      Atl. Div.
           $120.5K
           Pac. Div._
```

After ESC I ESC I, that becomes:

```
Quotas:
      Atl. Div.
           $120.5K
      Pac. Div.
```

Another ESC I returns you to the state of the last example, with P under the dollar sign. What happens if you try to reindent then, with no second word above? The words Pac. Div. go back to the left margin, since Emacs does not "know" about any further levels of indentation. However, if it did, it would indent to them:




Although no second word appears on the previous line, the line above that had one, so Emacs indents to that column. You can always "try" different levels of indentation by experimenting with ESC I and ESC 1 ESC I.

SUMMARY OF TERMS

- fill mode
- fill column
- fill prefix
- extended request

SUMMARY OF REQUESTS

| | |
|---|---|
| ESC X | extended-command |
| (ESC X) | fillon |
| (ESC X) | filloff |
| X. | set-fill-prefix |
| XF | set-fill-column |
| (ESC X) | opt paragraph-definiton-type |
| ESC Q | runoff-fill-paragraph |
| O | open-space |
| ESC I | indent-relative |

# SECTION 7

# SEARCHES AND SUBSTITUTIONS

Searching is the term applied to the editor's method of locating a particular character sequence, or character string. You may want to do this simply to move the cursor to a portion of your text where some editing is required, or you may be seeking to replace a character string with something else. This latter replacement of one character string with another is termed substitution. Usually, the easiest way to make a substitution is to search for the string to be replaced, delete it when found, and type in the correction. However, occasionally you wish to replace a particular string throughout the buffer; the substitution requests serve that purpose.

## SEARCHING FOR A CHARACTER STRING

Emacs provides two requests just for locating a character string, which you are prompted for. One of these searches forward from the current point to the end of the buffer. The other searches backward from the current point to the beginning of the buffer.

### ^S AND ^R

The ^S request is named string-search, and searches forward. When you type it, you are prompted:

```
String Search: _
```

Type in the characters you wish located, exactly as you expect them to appear, including any spaces where required, and terminate the prompt with CR. If the search succeeds, the point is left immediately after the first occurrence of the character string. If no such string is found, the cursor does not move and Emacs responds in the minibuffer:

```
Search fails.
```

Responding to a search request's prompt with only a carriage return reuses the last search string supplied (the default search string). Thus, you can locate several (or all) occurrences of the same string without continually retyping it.

The ^R request, reverse-string-search, searches backward through the buffer. It works just like ^S, except it leaves the point in front of the located string (so you do not keep locating the same string). Also, its prompt appears as:

```
Reverse String Search: _
```

## SUBSTITUTING ONE CHARACTER STRING FOR ANOTHER

These next two requests search for a specified character string, which you are prompted for, and replace it with another, also prompted for.

### ESC X replace and ESC %

The ESC X replace extended request substitutes one string for another at all occurrences of the first string between the current point and the buffer's end. You are prompted for the string to be replaced, and then for the string that is to replace it. If no occurrences of the first string are found, the cursor does not move. Otherwise, the point is left after the last substituted string.

The ESC % request, query-replace, allows you to substitute one string for another selectively. You are prompted for the search string and the replacement string individually; end each prompt with CR. This request searches forward for the first string, puts the point after it, and waits for one of the following responses (type the appropriate keys):

space

replaces this occurrence of the first string with the second. Then searches for the next occurrence of the first string and waits for a response again.

CR

leaves this occurrence of the first string unchanged and searches for the next occurrence of the first string, again waiting for a response after locating it.

. (period)

replaces this occurrence of the first string with the second and then terminates the query replace.

^G

leaves this occurrence of the first string unchanged and terminates the query replace.

SUMMARY OF REQUESTS

| | |
|---|---|
| ^S | string-search |
| ^R | reverse-string-search |
| (ESC X) | replace |
| ESC % | query-replace |

# SECTION 8

# TYPING SHORTCUTS

Several requests described in this section make the job of entering text a little easier. Included are requests for capitalizing words, underlining words, and transposing characters.

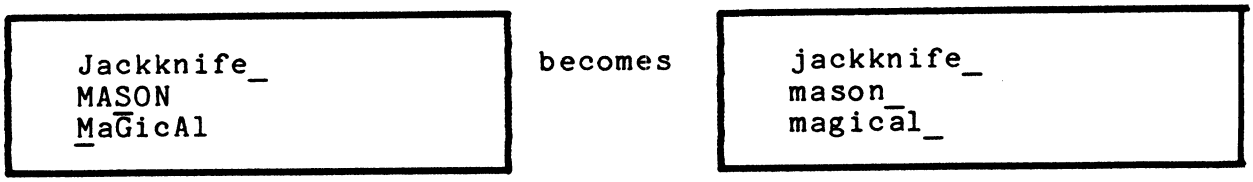
## CHANGING THE CASE OF WORDS

### ESC L, ESC U, ESC C

Three requests are provided for changing the case of a word to lowercase (jack), uppercase (JACK), or capitalized initial letter (Jack). Each can be issued when the cursor is on any character of, or immediately after, the word to be affected. If you issue these requests when the cursor is between words, but not immediately after the first word, they affect the second word. Each leaves the cursor immediately after the word. The requests are:

- ESC L, lower-case-word, to lowercase a word.
- ESC U, upper-case-word, to uppercase a word.
- ESC C, capitalize-initial-word, to capitalize the initial letter of a word.

Some examples of ESC L:



Examples of ESC U:

| | | |
|---|---|---|
| Friday<br>MON., tue. | becomes | FRIDAY<br>MON., TUE. |

Examples of ESC C:

| | | |
|---|---|---|
| president<br>EMACS<br>COngress_ | becomes | President_<br>Emacs_<br>Congress_ |

To change the case of several words in a row, you can issue ^Fs after each case-altering request to position the cursor for the next one:

```
thomas alva edison
```

after ESC C ^F becomes:

```
Thomas alva edison
```

Repeat the ESC C, ^F sequence:

```
Thomas Alva edison
```

A final ESC C:

```
Thomas Alva Edison_
```

## UNDERLINING

Related to the word case-altering requests are the word underlining and underline-removing requests. They underline a word, or remove the underlining from an underlined word. Since almost no video terminal can overprint characters, underlined text in Emacs appears as:

```
S\010__\010m_\010i_\010l_\010e
```

The \010s represent the backspaces. The text in your buffer, though its appearance is disconcerting, is written out with the proper number and placement of backspaces. The above would generally be dprinted as Smile. To avoid problems, never use backspaces for underlining; use ESC _ instead.

### ESC _ AND ^Z_

The ESC _ request, underline-word, underlines a word when issued while the cursor is at any character in the word, or immediately after it. When the cursor is between words, but not immediately after the first word, ESC _ underlines the second word (i.e., it affects the same word that the case-altering requests do). The cursor is left immediately after the underlined word:

```
hello
     -
```

becomes

```
_\010h_\010e_\010l_\010l_\010o
                              -
```

The ^Z_ request, remove-underlining-from-word, removes underlining from a word when the cursor is at any character in the word, or immediately after it. Again, when the cursor is between words, but not immediately after the first, ^Z_ removes the underlining from the second word. This request also leaves the cursor after the affected word:

```
_\010n010o_\010w
   -
```

becomes

```
now
   -
```

## TRANSPOSING CHARACTERS

^T

One of the commonest typing errors is to transpose characters. The ^T request, twiddle-chars, transposes (interchanges) the two characters to the left of the point. Generally, this means the two characters just typed. Thus, after a ^T:


SUMMARY OF REQUESTS

| | |
|---|---|
| ESC L | lower-case-word |
| ESC U | upper-case-word |
| ESC C | capitalize-initial-word |
| ESC _ | underline-word |
| ^Z_ | remove-underlining-from-word |
| ^T | twiddle-chars |

# SECTION 9

# HELP

Emacs has several requests whose only purpose in life is to respond to your appeal for help. They provide online descriptions of all the Emacs requests, i.e., you can find the request you need, or check to see how a particular request works, simply by issuing the appropriate help request.

## ASKING FOR A REQUEST'S DESCRIPTION

### ESC ?

When you are not sure what a certain key does, the ESC ? request, describe-key, tells you. First, it prompts you for the key you want explained:

```
Explain Key: _
```

Type the key sequence you want to find out about, for example, ^X^S, (no carriage return is required to end this prompt response). The minibuffer reads:

```
Explain Key:  ^X(prefix char): ^S
```

The description of the request appears as a local display:

```
^X^S                                    save-same-file
Save file.  Writes the contents of the current buffer to
its default file.  This request is equivalent to ^X^WCR.

--* * * * * * * * * * * *--
```

If you give ESC ? a numeric argument, e.g. ESC 1 ESC ?, you get just the key and its command-name, with no description. This saves time when you just want a quick check, since the command-name alone often provides the clue you want. With a numeric argument, ESC ? displays the information in the minibuffer instead of as a local display. Below is an example of the minibuffer's appearance with ^W as the prompt response:

```
Show Key Function:  ^W
^W = wipe-region
```

## ^_

The ^_ request, help-on-tap, provides several forms of help, depending on the character you type after the ^_ (on some terminals, you have to type ^? to send the ^_ character). If you type ^_? (or ^?? in those cases just noted), for example, you get a display of the current repertoire of ^_. In that repertoire are ^_D, ^_A, and ^_L.

The ^_D request gives you the descriptions of extended requests, just as ESC ? gives descriptions of key-sequence requests. It prompts you in the minibuffer for the command-name of the request to be described:

```
Extended command to describe:
HELP:  (? for more info):  _
```

(The second line above appears as soon as you type ^_. The prompt line appears above it when you type the "d".) You then type your response, e.g., "fillon", and a description of fillon appears in a local display.

The ^_A request lists all requests and extended requests that contain a given character string in their command-names, and tells what, if any, keys invoke them in the current buffer. You are prompted for the character string, and type it in:

```
String to match for apropos command:  forw_
```

As soon as you type the carriage return, the requests are listed, in this case, forward-word, forward-char, etc. Since this manual does not cover all of the requests, be prepared to see some command-names you do not recognize. You can use ESC ? or ^_D to find out more about any unfamiliar ones.

The ^_D request is equivalent to the ESC X describe extended request; the command-name to be described is supplied as an argument separated by a space from describe, before terminating the prompt with CR. The ^_A request is equivalent to the ESC X apropos extended requests; again, after typing apropos, type a space and then the match string before ending the prompt with CR.

The ^_L request gives a local display of the last 50 characters you typed. When unexpected things happen, you can issue this request to examine what you did so that you can identify and correct any problems.

## LISTING THE EMACS REQUESTS

### ESC X make-wall-chart

The ESC X make-wall-chart extended request puts a list of all the currently defined Fundamental mode command-names and their associated keys into a new buffer.S You can then write the contents out to a file and dprint a copy. The dprint makes a convenient wall chart for hanging near your terminal. The wall chart is divided into three columns (the dprint is 132 characters wide); a sample (of one column only) appears below:

| | |
|---|---|
| ESC F | forward-word |
| ESC G | go-to-line-number |
| ESC H | mark-paragraph |
| ESC I | indent-relative |
| ESC K | kill-to-end-of-sentence |
| ESC L | lower-case-word |

Figure 9-1. Sample Column of the Wall Chart

This request provides you with the tool needed to learn all the Fundamental mode requests. You can highlight selected requests on the chart so that you can refer to them at a glance when necessary. You can also experiment with unfamiliar requests that interest you, after obtaining their descriptions via the ESC ? and ^_D requests. In addition, by editing the wall chart before dprinting it, you can reorganize it to suit your own convenience. You may wish to group related requests under special headings, add notes that help you remember how they operate, or segregate seldom-used requests.

The wall chart lists many requests with which you are unfamiliar. In exploring them, you will be introduced to some of these Emacs features (described in the Emacs Text Editor Users' Guide):

- programming language major modes (alm-mode, pl1-mode, fortran-mode, lisp-mode, electric-pl1-mode) for convenience in writing and editing programs in those languages.
- mail facility (^XM, ^XR) the RMAIL major mode, for sending and receiving electronic mail.
- message facility (accept-messages) for handling interactive terminal messages.
- macro requests and Macro Edit major mode (edit-macros) for writing, editing and executing your own keyboard macros (a macro is a collected sequence of requests that automatically performs the whole sequence each time you execute it).
- multiple windows for displaying on the screen several buffers at once, and editing them.

- setting your own key bindings for requests and macros
- request recognizing and dealing with sentences and paragraphs
- requests for manipulating named marks and variables (regions)

When you enter one of the other major modes (e.g., via ^XM for RMAIL mode), you can issue the ESC X make-wall-chart extended request to list all the requests available in that mode. In addition to the Fundamental mode requests that still operate in the new mode, make-wall-chart lists the requests exclusive to the new mode. To find out more about these mode-specific requests, use the other help requests (ESC ? and ^_D) while in the relevant mode.

SUMMARY OF REQUESTS

| | |
|---|---|
| ESC ? | describe-key |
| ^_? | help-on-tap |
| ^_D | help-on-tap (describe) |
| ^_A | help-on-tap (apropos) |
| ^_L | help-on-tap |
| (ESC X) | make-wall-chart |

# APPENDIX A

## ALPHABETIZED LIST OF FUNDAMENTAL MODE REQUESTS

The following is a list of the Emacs Fundamental mode requests described in this manual, alphabetized by the last character. Everything preceding the last character of each request is arranged in this suborder: ^, ESC, ^X, ^X^, ^Z. Extended requests are listed separately at the end.

| | |
|---|---|
| # | rubout-char |
| ESC # | rubout-word |
| @ | kill-to-beginning-of-line |
| ^@ | set-or-pop-the-mark |
| ^X CR | eval-multics-command-line |
| delete key | rubout-char |
| ESC delete key | rubout-word |
| ESC | escape (for numeric arguments) |
| \ | escape-char |
| ^X. | set-fill-prefix |
| ^X= | linecounter |
| ESC % | query-replace |
| ^ | help-on-tap |
| ESC _ | underline-word |
| ^Z_ | remove-underlining-from-word |
| ESC < | go-to-beginning-of-buffer |
| ESC > | go-to-end-of-buffer |
| ESC ? | describe-key |
| ^A | go-to-beginning-of-line |

| | |
|---|---|
| ^B | backward-char |
| ESC B | backward-word |
| ^XB | select-buffer |
| ^X^B | list-buffers |
| | |
| ESC C | capitalize-initial-word |
| ^X^C | quit-the-editor |
| | |
| ^D | delete-char |
| ESC D | delete-word |
| | |
| ^E | go-to-end-of-line |
| ^X^E | comout-command |
| | |
| ^F | forward-char |
| ESC F | forward-word |
| ^XF | set-fill-column |
| ^X^F | find-file |
| | |
| ^G | command-quit |
| | |
| ESC I | indent-relative |
| ^XI | insert-file |
| | |
| ^K | kill-lines |
| | |
| ^L | redisplay-command |
| ESC L | lower-case-word |
| | |
| ^N | next-line-command |
| | |
| ^O | open-space |
| | |
| ^P | prev-line-command |
| | |
| ^Q | quote-char |
| ESC Q | runoff-fill-paragraph |
| | |
| ^R | reverse-string-search |
| ^X^R | read-file |
| | |
| ^S | string-search |
| ^X^S | save-same-file |
| | |
| ^T | twiddle-chars |

| | |
|---|---|
| ESC U | upper-case-word |
| ^V | next-screen |
| ESC V | prev-screen |
| ^W | wipe-region |
| ^X^W | write-file |
| ESC X | extended-command |
| ^X^X | exchange-point-and-mark |
| ^Y | yank |

EXTENDED REQUESTS

| | |
|---|---|
| ESC X | filloff |
| ESC X | fillon |
| ESC X | make-wall-chart |
| ESC X | opt paragraph-definition-type |
| ESC X | replace |

# APPENDIX B

## LIST OF FUNDAMENTAL MODE REQUESTS BY FUNCTION

The following is a list of the Fundamental mode requests described in this manual, grouped according to the functions they perform.

### MOVING THE CURSOR

#### Up or Down a Line

| | |
|---|---|
| ^P | prev-line-command |
| ^N | next-line-command |

#### Forward or Backward a Character/Word

| | |
|---|---|
| ^B | backward-char |
| ^F | forward-char |
| ESC B | backward-word |
| ESC F | forward-word |

#### To Beginning or End of a Line/Buffer

| | |
|---|---|
| ^A | go-to-beginning-of-line |
| ^E | go-to-end-of-line |
| ESC < | go-to-beginning-of-buffer |
| ESC > | go-to-end-of-buffer |

#### To Next or Previous Screen

| | |
|---|---|
| ^V | next-screen |
| ESC V | prev-screen |

## NUMERIC ARGUMENTS, LINE NUMBER, ABORTING A REQUEST, AND EXITING THE EDITOR

| | |
|---|---|
| ESC | escape |
| ^X= | linecounter |
| ^G | command-quit |
| ^X^C | quit-the-editor |

## DELETING

### Characters

| | |
|---|---|
| # | rubout-char (left of point) |
| delete key | rubout-char (left of point) |
| ^D | delete-char (right of point) |

### Lines

| | |
|---|---|
| @ | kill-to-beginning-of-line |
| ^K | kill-lines |

### Words

| | |
|---|---|
| ESC # | rubout-word (left of point) |
| ESC delete key | rubout-word (left of point) |
| ESC D | delete-word (right of point) |

### Regions

| | |
|---|---|
| ^@ | set-or-pop-the-mark |
| ^X^X | exchange-point-and-mark |
| ^W | wipe-region |

### Retrieving Text

| | |
|---|---|
| ^Y | yank |

## LITERAL CHARACTER ENTRY

| | |
|---|---|
| ^Q | quote-char |
| \ | escape-char |

## FILES

### Writing Out

| | |
|---|---|
| ^X^W | write-file |
| ^X^S | save-same-file |

### Reading In

| | |
|---|---|
| ^X^F | find-file |
| ^X^R | read-file |
| ^XI | insert-file |

## EDITING MULTIPLE BUFFERS

| | |
|---|---|
| ^XB | select-buffer |
| ^X^B | list-buffers |

## EXECUTING A MULTICS COMMAND

| | |
|---|---|
| ^XCR | eval-multics-command-line (no output expected) |
| ^X^E | comout-command (output expected) |

## REDISPLAYING THE SCREEN

| | |
|---|---|
| ^L | redisplay-command |

## SPACING AND FORMATTING

| | |
|---|---|
| ^O | open-space |

### Fill Mode

ESC X extended-command
ESC X fillon
ESC X filloff

### Margins

| | |
|---|---|
| ^X. | set-fill-prefix |
| ^XF | set-fill-column |
| ESC Q | runoff-fill-paragraph |

ESC X opt paragraph-definition-type

## Indentation

| | |
|---|---|
| ESC I | indent-relative |

## SEARCHES AND SUBSTITUTIONS

| | |
|---|---|
| ^S | string-search |
| ^R | reverse-string-search |
| ESC X replace | |
| ESC % | query-replace |

## TYPING SHORTCUTS

### Changing the Case of Words

| | |
|---|---|
| ESC L | lower-case-word |
| ESC U | upper-case-word |
| ESC C | capitalize-initial-word |

### Underlining

| | |
|---|---|
| ESC _ | underline-word |
| ^Z_ | remove-underlining-from-word |

### Transposing Characters

| | |
|---|---|
| ^T | twiddle-chars |

## HELP

| | |
|---|---|
| ESC ? | describe-key |
| ^ | help-on-tap |
| ESC X make-wall-chart | |

# APPENDIX C

# GLOSSARY

BUFFER

a temporary work space in which you create and edit new text, or edit a copy of a file read into the buffer for editing. The buffer contents instantly reflect any changes or additions you make; however, to save these changes, you must write the buffer contents out to a file, since buffers are discarded when you exit Emacs.

BUFFER, MODIFIED

a buffer whose contents have been changed in any way since the last time they were written out to a file. An asterisk appears at the beginning of the buffer's path line when it is modified.

BUFFER NAME

a name assigned to a buffer by you or Emacs. When a file is read in, the buffer takes its name from the first component of the entry portion of the file's pathname. When you first enter Emacs without reading a file in, the first buffer is named "main." When you create a new buffer via ^XB or reading a file in a second time, you assign any name you choose.

COMMAND-NAME

the name of a request. The name specifies to Emacs the set of programmed instructions to follow when the request is issued. For example, the command-name for the request that moves the cursor forward a character is forward-char.

CONTROL CHARACTER

a character typed while the control key is held down. Control characters are interpreted as requests to Emacs. Throughout the manual, control characters are represented by preceding them with the caret character (for the control key) and are depicted as uppercase (where applicable), e.g., ^F for control f.

CURRENT LINE

the line of the current buffer in which the cursor appears.

CURSOR

the blinking or solid box or underscore on the screen that represents your current position in the buffer (see "point").

DEFAULT PATHNAME

the file pathname that Emacs uses, by default, when you do not provide one. The ^X^F, ^X^R, and ^X^W requests can set the default pathname for the current buffer.

EXTENDED REQUEST

see request, extended.

FILL COLUMN

the first column in which text is not to be placed when in fill mode or when formatting with ESC Q. The fill column serves as the right margin, and is set by ^XF.

FILL MODE

a minor mode in which lines are automatically broken to provide a ragged right margin determined by the fill column (set by ^XF). The fill prefix (set by ^X.) is also inserted automatically at the beginning of each line. Fill mode makes carriage returns unnecessary.

FILL PREFIX

a fixed set of characters and/or spaces inserted at the beginning of each line when you type a carriage return, or inserted automatically when in fill mode or when formatting with ESC Q. The fill prefix is set by ^X..

KEY BINDING

the specific association of a request's command-name with a key sequence. The key sequence is bound to the request so that you can issue the request just by typing the special keys, e.g., typing ^XB to issue the select-buffer request. For requests without key bindings, you must type ESC X (the key sequence bound to the extended-command request) and then the command-name of the request, e.g., ESC X make-wall-chart. Any request can be issued by the latter method, whether or not a key sequence is bound to it, e.g., typing ESC X select-buffer has the same effect as typing ^XB.

KILL MERGING

the merging of successively deleted text on the kill ring; one ^Y can retrieve it. To merge, text can be killed with the ESC #, ESC delete key, ESC D, ^K, and ^W requests.

KILL RING

a ten-slot "ring" in which deleted text is saved for retrieval by ^Y. The ring rotates each time a slot fills; when all ten slots are filled, the next rotation discards text from the first slot so that it can be refilled, etc. The ^Y request retrieves text from the first slot, or from a specified slot when a numeric argument is given.

LOCAL REQUEST

a display of information that temporarily displaces the portion of the buffer shown on your screen. Requests such as ESC ?, ^_, and ^X^B put up a local display, the bottom of which is designated by a line of dashes and asterisks. Type a linefeed to remove any local display.

MAJOR MODE

an Emacs mode of operation, having its own related set of requests and key bindings to facilitate the performance of a special task. Fundamental major mode is used for general editing; separate programming language modes are used for writing and editing in specific programming languages; RMAIL mode is used for preparing and reading electronic mail, etc.

MARK

a fixed point set by ^@ to delineate one limit of a region; the cursor point delineates the other limit of the region. Up to ten marks can be set at any one time, the last one set being the current mark used by ^W and ^X^X. Using a numeric argument with ^@ moves the cursor to the mark specified by the argument.

MARK RING

a ten-slot ring, similar to the kill ring, for "remembering" the locations of the last ten marks set by ^@. The cursor can be moved to any mark on the ring by specifying the slot-number of that mark in a numeric argument to ^@.

MINIBUFFER

the two-line area below the mode line at the bottom of your screen, used by Emacs for messages and prompts so that they do not interfere with the text displayed in the window.

MINOR MODE

an Emacs mode of operation that modifies the way Emacs works in some specific area, e.g., fill mode provides a special formatting style. Minor modes, unlike major modes, do not have a unique set of key bindings.

MODE

see major mode, minor mode, and fill mode.

MODE LINE

a line immediately below the buffer's window that displays the Emacs major mode in effect in the current buffer (the mode name(s) is parenthesized), followed by the minor mode in effect, if any (enclosed in angle brackets), and the buffer's name (separated by a dash). For example:

```
Emacs (Fundamental <fill>) -- main
```

MODIFIED BUFFER

see buffer, modified.

NUMERIC ARGUMENT

an argument supplied for certain requests to alter the way they operate. Numeric arguments are given by pressing the escape key, and then an appropriate number. Most requests accepting numeric arguments simply repeat their action the specified number of times, e.g., ESC 4 ^F moves forward four characters. Some, like ^Y, ^@, and ESC I behave altogether differently when given a numeric argument.

PATH LINE

a line below the mode line that displays the absolute pathname (the default pathname) of the file in the current buffer. Until you specifically read a file into the buffer, or write the buffer contents out to a file, no path line exists. When you modify the buffer, as asterisk appears at the beginning of the path line, and remains until you write the changed file out again.

POINT

the position between characters indicated by the cursor's left edge. Editing actions always take place at the current point.

PROMPT

any phrase Emacs displays that requires a particular response from you before Emacs can continue action; called a prompt because Emacs is <u>prompting</u> you for the response.

REGION

an extent of text between the current point and the current mark, which has been set by the latest ^@. A region can be deleted by the ^W request.

REQUEST, EXTENDED

a request, usually with no key binding, that is issued by typing first the ESC X request (extended-command), and then the command-name in response to ESC X's prompt. Any request can be issued as an extended request when you can remember only its command-name and not its key sequence.

REQUEST

a set of programmed instructions to Emacs, identified by its command-name or key binding, if it has one. When you issue a request, Emacs executes that request's program, performing whatever editing procedure it defines.

SCREEN

the portion of your terminal used for the video display. Occasionally, "screen" is used synonymously for "window", which is actually only a part of the screen.

SUCCESSFUL KILL REQUESTS

a series of deleting requests that is uninterrupted by any other request. The text deleted by successive kill requests is merged on the kill ring; an ^Y retrieves it all.

WINDOW

the part of your terminal screen that displays a buffer's contents. Generally, this is the area between the screen top and the mode line.

# INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

CUT ALONG LINE

TITLE: LEVEL 68
INTRODUCTION TO EMACS
TEXT EDITOR

ORDER NO. CP31-00

DATED MARCH 1981

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

DATE _____

FROM: NAME _____

TITLE _____

COMPANY _____

ADDRESS _____

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

**HONEYWELL INFORMATION SYSTEMS**
**200 SMITH STREET**
**WALTHAM, MA 02154**

**ATTN: PUBLICATIONS, MS486**

**Honeywell**

CUT ALONG LINE

FOLD ALONG LINE

FOLD ALONG LINE

Honeywell

**Honeywell Information Systems**
In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5
In the U.K.: Great West Road, Brentford, Middlesex TW8 9DH
In Australia: 124 Walker Street, North Sydney, N.S.W. 2060
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

31091, 7.5C481, Printed in U.S.A.

CP31-00