1542

Bell Laboratories Cover Sheet for Technical Memorandum

The information contained herein is for the use of employees of Bell Laboratories and is not for publication (see GEI 13.9-3)

Title: An Interactive Screen Editor for UNIX

Date: June 16, 1980

TM: 80-5343-2

Other Keywords:


| Author(s) | Location | Extension |
|---|---|---|
| Warren Montgomery | IH 6E-314 | 2494 |


Charging Case: 49358-003
Filing Case: 40309-300

## ABSTRACT


High speed data communication and display screen terminals make possible a fundamentally different mode of entering and editing text to a computer system. An interactive screen editor allows a user to enter and edit files, and to see the effects of the editing immediately.

A good screen editor can improve productivity in several ways. The need for paper listings, and thus the expense and delay in dealing with them, is greatly reduced with the availability of a screen editor. The editor can provide a customized environment for particular tasks, such as editing program source or word processing, which can relieve the user of the mechanical parts of the task (such as maintaining proper indentation). The immediate feedback provided reduces mistakes, and speeds up their detection. A simple set of editing commands can be used effectively by relatively unskilled users, because of the feedback obtained by seeing the effects of editing.

This memorandum describes an interactive screen editor for UNIX known as EMACS. The editor is patterned after a very popular screen editor originally developed at M.I.T. It was developed by the author as a tool for his own work and is currently used by members of department 5343. The editor provides a friendly editing environment with the advantages outlined above, while running in the small address space provided to the UNIX user on the PDP 11/70. This memorandum describes the commands and editing environment of EMACS, and some experience with its use.


Pages Text: Other: Total:

No. Figures: No. Tables: No. Refs.:

Copy to
All Supervision Lab 534
All Members Department 5343
All Department Heads Lab 364
S. A. Bauman
C. Christensen
W. E. Danielson
J. D. De Treville
J. R. Fleming
J. A. Githens
R. D. Gordon
H. M. Jackson II
R. K. Kanodia
L. E. McMahon
D. M. McIlroy
E. Nussbaum
R. A. Reed
D. Ritchie
R. F. Rosin
J. M. Scanlon
K. Thompson
R. A. Thompson

CONTENTS

Bell Laboratories

subject: An Interactive Screen Editor for UNIX
Case: 49358-003
File: 40309-300

date: June 16, 1980

from: Warren Montgomery
IH 5343
6E-314 x2494
80-5343-2

TM: 80-5343-2

MEMORANDUM FOR FILE

## 1. INTRODUCTION

Text editing is the most common task of many computer users. The creation and modification of programs, data bases, and memoranda occupies much of the time that a user spends with a computer system. The ability to edit programs and recompile them was one of the primary reasons for the success of early time-sharing systems over batch processing systems.

Many of the text editing tools now in use are based on the editors for the early timesharing systems. These editors were developed for an environment that included mostly low-speed (110 baud) printing terminals, and expensive computer systems that were not prepared to interact with the user on a character at a time basis. In such a environment, it was appropriate to minimize the amount of output produced by the editor, and to allow the user to specify a lot of changes to be made by a single editor "command". Editors such as the standard UNIX editor (ed) are ideal for this environment.

In recent years, printing terminals have been repaced by display terminals capable of handling high data rates in many applications. The cost of computing has steadily dropped. The text editing tools made available to users must evolve to take advantage of these changes. With a high-speed display terminal, minimizing output is no longer appropriate. Instead, the display can be used to provide feedback for the user on the results of editing. The lower cost of computing, and better hardware support for terminals, make character at a time interaction with the computing system feasible. The EMACS editor described in this report is one attempt to take advantage of these effects in order to provide the user with a simple and powerful editing environment.

EMACS is a screen editor that can be used to build or to edit files using a display terminal, such as an ADM3A or SOROC. The

user interface to this editor is quite simple. The user is presented with a display of the contents of a portion of the buffer being edited. This display indicates exactly what is in the area being displayed, including any non-printing characters. The contents of the buffer being edited can be read from or written to a UNIX file. Characters typed by the user will be inserted into the buffer (and reflected in the display) at the point indicated by the terminal's cursor. This is the primary mechanism for entering and modifying text.

Control characters and escape sequences can be used to perform other editing functions, such as moving the cursor to a different position in the buffer, deleting text, replacing text, or searching. Thus there is only one mode of interpretation of characters typed to EMACS, in which either text to be entered or commands can be entered. This simple interface relieves the user of the need to remember what mode he is in, and prevents the disastrous mistakes that can occur when text to be inserted is evaluated as an editor command. A simple mechanism is provided to allow a user to insert control and escape characters when needed.

Although there is a rich vocabulary of commands available, including commands that perform functions tailored to a particular application (such as indenting a C program), the most common way in which EMACS is used to edit is simply to position the cursor over the area to be changed, and enter the changes. The immediate feedback provided by the visual display appears to be very important to the user.

This editor was written by the author as an aide to his other work, and patterned after the EMACS editor written for the PDP-10 systems at the M.I.T. Laboratory for Computer Science. The interface to the user closely follows that provided by the M.I.T. version, because the author was familiar with that version. The implementation of EMACS for UNIX described in this report was done by the author, and is entirely different from that used at M.I.T. The author was not familiar with the implementation techniques used in the M.I.T. version, only with the language in which EMACS was implemented.

The author and his organization are not supporting EMACS. The author is, however, willing to distribute copies of the software for use within Bell Laboratories, and is interested in comments regarding features or problems with EMACS. The author will repair problems as time allows, but makes no guarantees to fix problems promptly.

The remainder of this report contains a user's manual for the EMACS editor, and a discussion of the experience that we have had with EMACS in our department. EMACS continues to evolve to provide more commands and remove implementation restrictions. The users manual here describes EMACS version 3.0, which was in use in late May, 1980.

The display screen contains a window of approximately 20 lines into the buffer being edited. The terminal cursor is positioned at the point where the editor cursor is in the buffer. Each line of the buffer (delimited by a newline character) begins at the beginning of a display line. A line that exceeds the screen width is continued on the next screen line. Whenever a line must be continued on the next screen line an exclamation mark (!) is displayed in the last collumn of the first screen line. If the editor is in line number (lnumb) mode, then a line number is printed at the beginning of each line in the buffer.

Printable characters are displayed normally, while tabs are displayed as white space up to the next screen collumn that is a multiple of eight. Each non-printing control character is displayed as a two character sequence of '^' followed by the control character + 100 octal. This causes a control-x to display as '^X'. This mapping produces a reasonable display for most of the control characters. There are several other non-printing characters for which the character displayed is not obvious. Rubout displays as ^?. The "us" character (037) displays as ^_. The "rs" character (036) displays as ^^. The "gs" character (035) displays as ^]. The "fs" character (034) displays as ^\, and the escape character (033) displays as ^[. A character which has the high order bit set (0200 octal) is displayed with an M- in front of it. This bit is normally not set in ASCII characters.

In addition to the display buffer, several lines of the screen are used for status information and for displaying parameters entered into EMACS, such a file name. One of these lines known as the status line contains the editor name, editor version, buffer number and name, and file name. Some of the more recently introduced commands described in this document indicate the version in which they were introduced, so that you can determine whether or not a particular command is in the version that you are running. If the buffer has not been modified since the file was read or written, an '=' will be displayed between the buffer and file names. Otherwise, a '>' will appear.

The lines below the status line are used for the time of day display (if time mode is on), and for emacs to prompt for parameters for commands. Some commands cause the buffer display to be erased in order to display other information in place of the buffer. The word "Continue?" will be displayed at the bottom of the screen when this happens. Typing 'y', ' ', or return will bring back the buffer display. Typing 'n' may allow you to re execute the command producing the display.

Figure 1 shows a typical screen during a EMACS session. The buffer "Main", number 0, is being used to edit a program test.c. The buffer has been modified since the last write to the file test.c.

Figure 1 EMACS screen Display

```
1   #include <stdio.h>
2   /* EMACS_MODES: c, !fill, comcol=43 */
3
4
5   /* This is a c program */
6
7   main()
8   {
9           int i;
10          char c;
11
12          for (i = 0;i > 0; i++) {
13                  printf("i = %d\n",i); /* print i */
14          }
15  }
16

EMACS 2.8  (0) Main > test.c
```

## 2. EDITING WITH EMACS

As noted above, when the buffer is displayed by EMACS, one can enter characters into the buffer being edited simply by typing them on the keyboard. Editing functions use control characters or escape sequences. This section provides a brief description of the basic editing commands available, and the way in which they are invoked.

### 2.1 The Character Set

EMACS operates on characters from an alphabet of 256 different characters. These include the 128 ASCII characters that can be entered from a terminal, and 128 "Meta" characters. A Meta character is entered by preceding it with an escape (ESC key). In this document, and in characters displayed by EMACS, control characters are represented as a character preceded by '^', and Meta characters are represented as a character preceded by 'M-'. Thus the character 'M-a' (Meta a) can be entered by hitting the ESC key, followed by the a key. The character '^X' is the character obtained by hitting the control key and the x key.

Each character that is typed into EMACS is interpreted as a command. All of the ordinary printing characters insert themselves into the buffer being edited at the point defined by the cursor. Thus in order to enter text, simply type it. The control and meta characters are used for editing commands that manipulate the text in the buffer, or move the cursor, or both.

## 2.2 Arguments and Parameters

All commands, including the printing characters, take a numeric argument that has some effect on their interpretation. The default argument given to a command for which no argument is specified is 1. To specify some other argument to a command, you can enter escape, followed by a sequence of digits, and then the command. Numbers starting with a 0 are interpreted as octal, while numbers starting with any other digit are decimal. A second way of specifying the argument is to precede the command by one or more ^U (control-u) characters. Each ^U multiplies the value of the argument by 4.

For most commands, the effect of the argument is to multiply the number of times that the command is applied. Thus the sequence ^U^Ux inserts 16 x's into the buffer at the current location. The sequence ESC13^N moves forward 13 lines in the buffer.

In addition to the numeric argument given to all commands, some commands will prompt the user for additional character string parameters. The commands that take parameters, and the method of entering parameters are described in the section on file and buffer commands.

## 2.3 Simple cursor movement commands

There are many ways to move the cursor around in the buffer without modifying the text in the buffer. Most of these use their argument to specify how many times the movement is to be repeated. These commands include:

^F — Move forward one character. On reaching the end of a line, the cursor moves to the first position of the next line.

^B — Move backward one character. On reaching the beginning of a line, the cursor moves to the end of the previous line.

^N — Move down one line. The cursor is moved to the same character position in the next higher number line in the buffer. Note that if the buffer contains tab or control characters, the same character position in the lines in the buffer may display at different points in the screen.

^P — Move up one line. The cursor is moved to the same character position in the next lower number line number in the buffer.

^A — Move to the beginning of the current line of the buffer. Note that both this and the following command work on the current line in the buffer, which may be

displayed on two or more screen lines if it is too long to fit on one screen line.

^E Move to the end of the current line.

M-< Move the cursor to the beginning of the buffer.

M-> Move the cursor to the end of the buffer.

M-f Move the cursor forward one word. Note that words are delimited by non- alphabetic or non-numeric characters.

M-b Move the cursor backward one word.

M-g Move the cursor to the line number specified by the argument given to the command.

^V Move to next page. The cursor is moved forward so that a new window, beginning with the first line not currently displayed, will be displayed.

M-v Move to previous page.

## 2.4 Inserting 'odd' characters

Because EMACS uses control and escape characters for commands, you cannot directly insert them into the buffer by typing them. The following three commands are useful for the occasional need to get such characters into a buffer.

^Q Quote the next character(s). ^Q accepts one or more characters (the number of characters specified by its argument) from the terminal and inserts them "blindly" into the buffer without interpretation. Only the new-line (line feed) character is interpreted. EMACS strips the parity bit from all characters read from the terminal, so all characters inserted this way have zero parity.

M-q Quote characters and turn on parity bit. This acts just like ^Q, however it turns on the parity bit in the character before inserting. Characters inserted this way will be displayed as meta characters by EMACS.

M-\ Convert the argument to a character. This command takes its argument and converts it to a character and inserts it. This provides an easy way to convert from octal or decimal to ASCII, and is occasionally useful for creating "funny" character strings.

## 2.5 Text Deleting commands

Several commands are available to delete text from the buffer. All of these commands operate on text near the current cursor position. The deletion commands are:

^? (rubout) Delete the character before the cursor. The character before the cursor usually the last character typed. A rubout deletes it.

^H (backspace) Backspace is a synonym for rubout, also deleting the previous character.

^D Delete the character under the cursor. If a newline is deleted (rubout at the beginning of the line or ^D at the end), lines are joined.

^K Delete to the end of this line. If invoked with an argument of 1, ^K deletes the remaining text on this line (if any). If no text follows the cursor on the current line, ^K deletes the newline. If ^K is given an argument greater than 1, it deletes multiple lines (including the newlines).

M- (Meta space) The command Meta space or ^@ (control @) places an invisible mark on the current cursor position. This mark can be used in subsequent editing. EMACS maintains 12 marks, one for each buffer. If an argument is specified to this or to other commands that use marks, it is used to select which mark to use. If no argument is given, the buffer number is used to select a mark. Thus ordinarily, EMACS acts is if there is one mark in each buffer, however you can use as many as 12 marks in a single buffer by explicitly specifying which mark to use. Each mark is simply a position in the buffer (line number and character within the line.) Thus if you add or delete text in front of a position where a mark was placed, the mark may not remain on the same character, but stays on the same position.

^W The command '^W' deletes the text between the current cursor position and the mark. This is a convenient way to delete a well defined block of text. If an argument is specified, it is used to select the mark number The mark can be either before or after the cursor position and achieve the same effect.

M-^? (Meta rubout). This command deletes the previous word of text. The definition of a word is the same as for M-f and M-b

M-d This command deletes the next word of text.

## 2.6 File and buffer Commands

Most of the file accessing commands are invoked through the ^X command. ^X is a prefix for several useful commands, most of which involve file or buffer access. These commands are invoked by a ^X followed by a second character.

Many of these commands prompt for parameters, such as a buffer name or file name. With all of these commands, you can enter text, just like you enter it into a buffer. Rubout (or backspace) deletes the last character entered and @ or ^K deletes the entire entry and lets you start over again. ^Y enters the current file name (sometimes useful), and ^X enters the current line from the buffer. ^G aborts the command without doing anything. Escape, newline or return terminate the parameter being entered.

^X^R Read file. EMACS will prompt for a file name, which you enter as described above. When the file name for ^X^R has been entered, EMACS will read the specified file into the buffer. If you invoke ^X^R with an argument of one (the default argument), it clears the buffer before reading. If you invoke ^X^R with an argument that is not 1 (i.e. ^U^X^R) it inserts the file into the buffer at the current cursor position. If the cursor is in the middle of a line, reading will be much slower, so it is best to open up a blank line when reading a file into the middle of a buffer, and then delete any unwanted lines after the file is read.

^X^W Write file. Prompting for file name is as above.

^X^S Save file. This writes out the buffer to the last file read or written if the file has been modified.

^X^B Change buffer. EMACS allows up to 12 named buffers to be edited concurrently. '^X^B' prompts for the name of a buffer, and makes that buffer the current buffer. If the buffer name "..." is entered, a new, empty buffer with a unique name is created. If a null buffer name is given (just type newline when asked for buffer name) a list of the active buffers is displayed. Typing 'n' in response will cause EMACS to ask for the buffer name again, while typing anything else will return you to the buffer display. All commands that ask for a buffer name will also accept a buffer number. The buffer number is displayed in parentheses next to the buffer name on the status line.

^X^F Find file. This command prompts for a file name and switches to a buffer that holds the specified file. If the specified file has been read into a buffer, the effect of find file is to change to that buffer. If no

buffer holds the specified file, the effect of find file is to create a new buffer and read the specified file into it. Find file is a convenient way to switch between editing several files.

^X^X Exchange the cursor position and the mark. An argument can be specified to indicate the mark to exchange with.

^X^I Re-direct input. This command directs EMACS to take input from a file. The file is assumed to contain EMACS commands, and can be created by editing with EMACS, using ^Q to enter control and escape characters. One use for this command is to perform a series of commands on the current buffer, or to set up a standard set of modes. Note that if the file contains only printable ASCII text, tabs, and newlines, ^X^I will effectively read the file into the buffer at the current location. Note, however, that this is very slow, and much better done with ^X^R.

^X^C (or ^Z) Quit Emacs. If any buffers have been modified since the last write, EMACS will ask whether or not to write out each such buffer before exiting. EMACS will not ask whether or not to save an empty buffer or a buffer with a null file name.

^X^T Send text to another buffer. This command sends the text between the mark and the current cursor position in the current buffer to another buffer. EMACS prompts for the name of the other buffer, and the text is inserted into that buffer at the current cursor position for that buffer. The current buffer remains unchanged. If an argument is given, it selects the mark to use.

## 2.7 Miscelaneous commands

The above set of commands are sufficient to do a substantial amount of editing quite easily. Here are more commands for special situations.

^G Abort. Typing ^G at any point that emacs is asking for input (except ^Q and M-q) will abort the current command. This applies at any step (specifying arguments, typing escape, entering parameters that emacs asks for, etc.). This is a convenient way of aborting anything that you are not sure that you want to complete (or how you got there).

^X2 Enter Two window mode. In two window mode, the screen is split in half vertically, and each half displays a different buffer. One window is current, as indicated by the cursor position and the file name on the status

line, and one window is dormant, and continues to display the text last put there. Two window mode is a good way to keep a display of something like an include file containing definitions handy while editing another file. ^X2 will prompt for the name of the buffer to display in the second window, and move to that window. Once in two window mode, the buffer displayed in the current window can be changed by any of the buffer switching commands. You can have two windows on one buffer, but only one is "active", so that changes made are only reflected in the active window. Text in the dormant window is wiped out by any command that re-writes the screen with something other than the buffer, such as M-?.

^X1 Enter one window mode. Return to one window mode from two window mode

^X^O Switch windows. Make the dormant window current and the current window dormant.

^Y Insert last killed text. All text that is deleted is saved in a "kill stack". The kill stack can hold the last 8 deletions. There is also a limit on the total amount of text that can be held in the kill stack, but you are unlikely to encounter it. ^Y retrieves the most recently deleted text. The most frequent use of this command is in moving text around. The procedure is: kill the text to be moved, move the cursor to where you want it, and enter ^Y. Another use of ^Y is to undo an unwanted deletion. ^Y leaves the mark (the one corresponding to the buffer number) at the beginning of the inserted text, and puts the cursor at the end. Both ^Y and M-Y work much faster if the cursor is on a blank line. Thus if you are moving or retrieving lots of text, it is best to open up a blank line to do the retrieve to, and then delete any unwanted text when done.

M-Y Replace last retrieved text. This command kills the text between the cursor and the mark corresponding to the buffer number, and replaces it with the next to last item on the kill stack. This command only really makes sense when it is issued immediately after ^Y, where it can be used to change the text that has just been retrieved. By entering ^Y followed by some number of M-Y's, any text in the kill stack can be retrieved.

M-p Pickup the region of text. This command picks up the text between the current position and the mark and puts it in the kill stack, without changing the buffer. This is useful for duplicating blocks of text in the buffer. An argument can be used to specify which mark

to use

^O Open up a line. This command creates one or more empty lines at the current cursor position. This is useful for inserting text in the middle of the buffer, while minimizing the amount of screen refresh needed.

^T Transpose the next two characters.

^S Incremental search. EMACS will prompt for a search string and as the search string is entered, begin searching for the next occurrence. At any point, the cursor sits at the beginning of the text that was found, and the search string is displayed at the bottom of the screen. Rubout can be used to delete the last character in the search string. If no match is found, the terminal bell rings. A message indicating the failure is typed as the prompt. Entering ^S in place of another search character causes the search to proceed to the next occurrence of the search string. Entering ^R cause the search to proceed backwards. Entering either ^R or ^S with a null search string causes the last search string (from the last search) to be retrieved. Entering ^G quits the search and returns the cursor to its previous position. Entering ESC stops the search, leaving the cursor where the search took it. Entering any control character stops the search, and then interprets the command specified by the control character.

^R Reverse search --, like above, only starts backward.

M-R Query replace. You will be prompted for a From string and a To string. Each can be edited as the filenames above. In the To string, the '&' character can be used to designate replacement with the From string. To get a real '&', prefix it with a '\'. To get a real '\', prefix it with another '\'. EMACS then searches for the from string and positions the cursor in front of it. Typing ' ' (space) or 'y' causes that from string to be replaced by the to string and moves to the next occurrence of the from string. Typing rubout or 'n' moves to the next occurrence of the search string. Typing '^G' stops the replacing. Typing 'r' causes EMACS to replace all of the subsequent occurrences of the search string without asking. Query replace stops when the from string is no longer matched. '?' prints a summary of the options

M-^S Regular Expression Search (Version 2.1) This will prompt for a regular expression to search for. You can edit the expression like editing filenames. The syntax is standard UNIX regular expression syntax. The search

finds the next occurrence of the search expression, wrapping around at the end of buffer. Searching for expressions that span line boundaries will not work.

M-^R Regular expression query replace (Version 2.1) This is just like query replace, except that a regular expression is allowed in the search string.

M-/ Begin comment. This command begins a C program comment by moving to the appropriate collumn and putting a /* in the buffer. The next newline will close the comment and automatically append a */

M-! Unix escape. Prompts for and executes a unix command line. If this command is given an argument (i.e. ^UM-!), it passes the contents of the current buffer to the command as standard input.

M-$ Unix escape. This works just like the above, except that it also traps all of the standard output from the command executed and saves it in a buffer called .exec as well as sending it to the terminal. If you don't give M-$ an argument, the buffer .exec is cleared first, while if an argument is given (i.e. ^UM-$) the output is appended to whatever was in buffer .exec before. This is useful for saving a copy of the error messages produced by a C compilation of a file being edited, for example. The file name of the .exec buffer is set to the command line that produced it. This can be useful if you want to re-execute the same command, as you can make .exec your current buffer, enter M-$, and enter ^Y followed by newline as the command line. ^Y gets the old command line back, and newline will execute it.

M-^M Mail. This command takes the current buffer as unix mail, and sends it. The buffer must contain at least one line starting TO: , which specifies the recipients of the mail. Each recipient is delimited by a space. Any number of recipients may be listed in a single line, however to improve readability, additional TO: or CC: lines may be used in specifying lists of recipients. Any errors encountered by mail are printed.

M-_ Underline word. This command underlines the following word of text. Useful for generating underlined text for mm.

^C Capitalize. This command capitalizes the letter under the cursor and moves the cursor forward one position. Lower case alphabetic characters are converted to upper case, while other letters are unchanged.

M-c Capitalize word. The letter under the cursor is capitalized, and the cursor is moved to the beginning of the next word.

M-t set terminal type. Prompts for terminal type and sets the character sequences used to display text to be appropriate for that terminal. Available terminal types are soroc, adm, adm31, vt100 (ANSII mode, 132 columns), vt52 (vt52 or vt100 in vt52 mode), informer (64x16), and hp. EMACS uses terminal commands for relative and absolute cursor position, clear screen, clear from the cursor to the end of line, insert and delete lines, and insert and delete characters. The number of characters transmitted for re-display will depend on which of these functions are available. Thus EMACS transmits fewer characters on an adm-31, which has all of these functions, than with an adm-3a, which has only cursor positioning and clear screen. Most terminals fall between these extremes.

M-? Explain. This command prompts for a character and prints a brief explanation of what that character does.

M-w Wall Chart. This command puts a listing of all commands (including those prefixed by ^X) into the current buffer. This command is a convenient way of producing a "wall chart" of the commands. The list is inserted into the buffer at the current position, so that normally one would want to execute it in an empty buffer. The appendix to this report contains a current copy of the wall chart.

^L Refresh. refresh the display. Occasionally, some error may cause the display to become garbled. ^L recreates it from scratch.

M-^L Redisplay top. Redisplay the window with the current line at the top. This is useful for viewing the lines that follow the current line. Note that this does not re-create the entire display, as does ^L, so it will not correct garbling.

M-" Auto Fill Buffer. This command re-adjusts the lines in the buffer so that each line contains 72 or fewer characters. The adjustment is done like nroff (mm) by moving words from one line to another. nroff or mm command lines and blank lines are preserved as is. This command can be used to improve the way in which an nroff or mm source file displays, by getting rid of long lines, without affecting the output.

## 2.8 Macro Commands (version 3.0)

EMACS provides a facility to allow a user to define his own commands. These "macro commands" provide a way to implement specialized editing functions for particular applications. A macro command is actually a sequence of EMACS editing commands that are invoked in response to a single character entered from the keyboard. Macros can contain ordinary emacs editing commands, invocations of other macro commands, or one of the commands described in this section that are of little use in interactive editing, but are quite useful in macros.

Because there are a large number of commands related to defining and using macros, and because these are of limited interest to most users, macros are documented in a separate manual.

## 2.9 Modes

EMACS has a variety of modes and other parameters that can be changed from commands entered in the terminal. The current setting of modes can be displayed by typing ^X^M followed by a return. There are three types of modes: on/off modes, integer modes, and string modes. ^X^M displays the names of the on/off modes that are now on, and the values of the string and integer modes. The '^X^M' command can be used to set these parameters. '^X^M' will prompt for the name of the mode to set, and if it is an on/off mode, turn it on if the argument is 1, and off if the argument is anything else. (Thus mode is set to the value of the argument. For a string mode, EMACS prompts for the new value. The modes and types are listed below, along with their default values. For ON/OFF modes, the default is underlined.

save — Auto Save Mode (ON/OFF)
If auto save mode is on, EMACS will automatically write the current buffer after savetype characters have been entered since the last save. This mode reduces the chance of disaster in the event of a crash, but can be annoying by causing a lot of writing.

savetype — Save type ahead (INTEGER=256)
If auto save mode is on, this is the number of keystrokes between saves.

fill — Auto Fill Mode (ON/OFF)
If this mode is on, emacs will automatically move to the next line whenever the cursor moves to the end of the line, breaking the line at a word boundary. This is very useful for entering text, as no newlines need be entered in the middle of the text.

fillcol — Auto Fill Column (INTEGER=72)
This is the character position beyond which auto fill mode will cause the line to be broken.

lnumb — Line Number Mode (ON/OFF)
This mode causes the current line number to display at the left of each line.

c — C Mode (ON/OFF)
This mode automatically indents for a C source file. Each line is indented with the number of tabs in the last non-comment line plus the net excess { characters over } characters in the last line. This mode is particularly useful for entering c program text.

tabstop — Tabstop (INTEGER=8)
This mode is the number of characters per tab that are displayed. Displaying a deeply indented c program may look much better if tabstop is set to something smaller than the eight default.

comcol — Comment Collumn (INTEGER=40)
This is the collumn in which comments entered via M-/ begin.

backspace — Backspace Mode (ON/OFF).
Turning on backspace mode causes backspace characters (^H) to display as moving back one column rather than as a ^H. This is very useful for viewing runoff output, or manuals, but editing the resulting text can be a bit tricky, because it is impossible to tell whether the character under the cursor is the one being displayed, or one that has been overprinted, or a backspace.

time — Time mode (ON/OFF)
When time mode is on, EMACS will display the time of day below the mode line (the one that says EMACS and the buffer name). The time is updated every time a character is read. Using time mode when entering lots of text is expensive.

verbose — Verbose mode (ON/OFF)
When verbose mode is on, EMACS will prompt for more input when ^X, ^Q, escape, or ^U are entered. This makes it easier to keep track of where you are.

overwrite — Overwrite mode (ON/OFF)
In overwrite mode, text entered will overwrite text already there. Text entered when the cursor is at the end of a line will be inserted as

before. Overwrite mode may be more natural for some people when making corrections.

nobell — No Bell (ON/OFF)
Ordinarily, unexpected conditions, such as errors or quitting out of commands, cause the terminal bell to ring. On some adm3a terminals, This causes some kind of disaster to occur. Turning on nobell mode prevents EMACS from ringing the terminal bell.

You can specify the modes to be used while editing a particular file by putting the string "EMACS_MODES: " somewhere in the first 10 lines of the file. The text on the same line following EMACS_MODES: will be taken as names of modes to set on or off. A mode name preceded by '!' will be set off, while mode names just listed will be set on. If '=' immediately follows a mode name, then the characters immediately following the '=' will be taken as the value for the mode name. Any text on the line that does not correspond to a mode name will be ignored. Thus the line:

```
/* EMACS_MODES: !fill c, comcol=43 */
```

in a c source file will set c mode on, fill mode off, and set the collumn for starting comments to 43. These modes are set whenever the file is read, and whenever you switch buffers.

## 2.10 Getting started

When EMACS is invoked, it must find the terminal type of the terminal that you are using in order to know the appropriate escape sequences for moving the cursor and clearing parts of the screen. This is taken from the shell variable TERM. To make it work, you must set the value of $TERM in your .profile, and export it.

On the IHNSS UNIX system, we have a utility program called ttype that will print out the type of a terminal that is on a hard wired port. Thus the lines:

```
export HOME PATH MAIL LOGNAME LOGTTY TERM INFO
TERM=`ttype`
```

will set the terminal type correctly on the IHNSS UNIX system. If TERM is set to an unrecognizable type, EMACS will prompt for terminal type, however the prompting message may come out in a strange place on the screen due to lack of knowledge about terminal controls. The acceptable terminal types at this time are: adm (adm3a), adm31, soroc, vt100, hp, informer. Support for other terminals could easily be added if needed.

There are two "features" of the adm3a terminals that sometimes cause trouble in EMACS. One of the internal switches on that terminal is labeled space/advance. That switch must be in the

space position for EMACS to work properly. A second problem is that many of these terminals are set up to use one of the "undefined" pins of the rs-232 interface for some internal diagnostic. This causes the terminal to go wild every time it receives a bell character when connected to a 1200 baud modem (which uses the same pin for something else). If this happens to you in EMACS, set nobell mode, which will prevent EMACS from sending bell characters to your terminal. A better and more permanent solution is to modify the terminal to disconnect this lead, use a cable that does not carry the undefined signals between the modem and the terminal, or use a different kind of modem.

Once terminal type has been established, EMACS reads the file .emacs_init in your home directory as a set of start-up commands. You can put initializations of various modes to your liking in this file. If the file does not exist, it is not read, and you will get a default setting of modes (simple enough!).

## 2.11 Helpful hints

This editor is very easy to use, once you know a few of the basic commands. Learn a few of the basic commands at a time. EMACS tries to be reasonably efficient about the refreshing of the screen. Some sequences, however, will cause lots of text to be redisplayed. While you can insert anything into the middle of a buffer by typing it, it is much better to open up some lines where you want to insert, insert, and then kill unneeded lines.

Use M-? when in doubt about a command. The explanations are brief, but should be sufficient to tell you what you want to know. Like most editors, EMACS maintains a local buffer, so that changes made do not go into the file until the next write. Type ^X^S reasonably frequently so as to avoid being wiped out by machine crashes, editor bugs, or other unpredictable events.

Because EMACS tries to avoid unnecessary refreshing of the screen, it will get confused if characters are sent to your terminal from some other program while running EMACS, such as a "background" program, or by a write command from another user. If you suspect that the display does not correspond to the buffer that you are editing, type ^L to refresh the screen. After typing ^L, the screen will match the buffer being edited.

## 2.12 Limitations of the editor

There are some limits that you may encounter:

- Each line can hold at most 512 characters.
- Each buffer can hold a maximum of approximately 15,000 lines.

- You can have at most 12 buffers.
- The kill stack contains the 8 most recent deletions, or a total of 256K characters.
- Filenames are limited to 128 characters.

## 2.13 Problems with EMACS

Because EMACS puts your terminal in "raw" mode, there is no way to transmit a "quit" signal to EMACS if something goes wrong. (On a hard-wired port, you may have to kill the EMACS process from another terminal, as there may be no way to send a hang up signal). If EMACS encounters an error that causes a fault trap (memory fault, bus error, etc.) EMACS will put your terminal back in cooked mode before exiting. If EMACS is killed via the kill command (except kill -9), it will try to save your buffers before terminating. The buffers are saved in the files emacs0-emacs11 in your home directory. A message to this effect is printed, although it is easy to miss.

# 3. DIRED

The program DIRED (DIRectory EDitor) allows editing of a directory. DIRED takes an argument which is the name of a directory, and displays the result of a ls -al executed on that directory. The EMACS commands for cursor movement, search, etc. can be used to move around in the displayed buffer. Entering 'd', '^D', or '^K' marks the file designated by the current line for deletion, while entering 'u' removes such marks. All files marked for deletion are displayed with a D in the first column. Doing a save, write, or exit causes dired to print a list of files to be deleted, and ask for permission to delete. If yes is entered, DIRED will delete those files. Doing a read (^X^R) will read another directory for display without deleting any files.

Another feature of dired is that typing 'e' will cause the file or directory designated by the current line to be edited by DIRED or EMACS as appropriate. This is a convenient way of exploring the useful files in some directory. DIRED is a nice way to clean up a "dirty" directory, because it allows the user to go back and forth over the contents of a directory, marking files for deletion.

DIRED is not very robust, and can be confused if, for example, you try to delete directories, edit the filename field in the display lines, or delete the header line.

## 4. EXPERIENCE

The author originally wrote EMACS as an aide to his own work of programming and memorandum composition. EMACS was made available to other members of the author's department (5343) and now has a user community of about 15. The observations in the following sections come from the author's experiences and those of the other users.

### 4.1 Feature Use

The most commonly used "feature" of EMACS is simply the ability to enter text and see the result. The simple cursor movement commands appear to be much more frequently used than the more complicated commands, such as those moving forward or backward by sentences.

The following is a summary of the user reaction to some of the unique features of EMACS.

Multiple Buffers

The ability to maintain several editing buffers at the same time is widely used for many purposes. These include examining several source files while making modifications, holding source, compiler output, and program output while debugging, and holding the output of the spell utility while scanning a document for misspelled words. The availability of multiple buffers allows the user to manage several different tasks, such as writing a program, editing a memorandum, and helping a friend locate a system problem, concurrently.

Two Window Mode

Two window mode is less heavily used than some of the other features, but appears to be quite useful for a variety of tasks. These include viewing a buffer containing declarations at the same time that a program is being entered in the second window, and viewing the output of the spell program at the same time that the document is being edited. The fact that only one of the two displays is actively maintained does not seem to be a problem, nor does the limitation of only two display windows. The author has not received any user comments on these limitations.

Incremental Search

With a high speed terminal, incremental search seems to be very effective. The ability to quickly move from one occurrence of a string to the next seems to meet most user's needs. With significantly less than 9600 baud communication, the amount of re-display created by incremental search becomes bothersome, and the regular expression search is preferred. The fact that incremental search does not "wrap around", seems to be a significant problem to many users.

C mode and Fill mode

These two modes provide customized facilities for editing certain kinds of files. The user reaction to these is mixed. Some users make extensive use of these, while others have their own styles for entering memorandum or program source and do not use these modes. They are a very small part of EMACS, and as such are worthwhile even if they are not extensively used.

Line Numbers

The display of the line number (lnumb mode) is one aspect of EMACS that was not taken from the M.I.T. version. This appears to be extremely useful for use with UNIX, primarily because of the widespread use of line numbers to indicate position within a file. Line numbers also serve to provide the user with some feedback as to what part of the buffer is currently being displayed.

## 4.2 Performance

While computer resources are not as expensive as they once were, they are far from free. An interactive screen editor such as EMACS consumes more in computer resources than a conventional editor. EMACS was designed to be as efficient as possible, some price must be paid for the benefits gained.

The Processor time consumption of EMACS is comparable to the UNIX editor for most "commands", including reading and writing files, searching, and substitution. The major discrepancy in performance comes in entering text. In entering text, EMACS consumes approximately 2.5 times as much processor time as the UNIX editor.

If we examine how each editor uses its time, we see that almost all of the time spent by the UNIX editor is spent in the UNIX operating system reading characters. About two thirds of the time spent in EMACS is spent reading and writing, while the remaining third is spent re-creating the display. EMACS uses almost twice as much UNIX operating system processor time, because it is re-writing the display after each character is typed, in addition to reading the characters. In addition, EMACS spends a significant amount of user processor figuring out how to most efficiently re-display the screen.

While EMACS does consume more time editing than does the UNIX editor, the processor time spent editing does not appear to be a significant problem. Based on the use of EMACS in our department, about .01 seconds of PDP 11/70 processor time are consumed for each character input to EMACS. This is an average over all users doing many different kinds of editing tasks. There is substantial variance from one case to another. This rate suggests that a substantial number of users could be supported by a PDP 11/70. In our department, we typically have around 8 users

running EMACS during the afternoon, with no significant complaints about the response that can be attributed to EMACS.

## 5. CONCLUSIONS

The EMACS editor provides an effective means of using a high-speed display terminal for text editing. EMACS provides a variety of unique features that seem to be very useful in editing. User reaction indicates that using EMACS has improved their productivity, however there are no quantitative measurements of this effect.

EMACS uses more computing resources than the standard UNIX editor, however the resources utilized do not appear to be a serious problem. We have not yet experienced any performance problems attributable to the widespread use of EMACS on the PDP 11/70 used by our department.

EMACS was written by the author as a tool for his own work, but is available to anyone desiring a copy. There are no currently known bugs, however there may be undiscovered problems. The author is interested in user reaction to EMACS and in reports of problems, however, the author's job does not include supporting EMACS, and thus the author does not promise any prompt response to suggestions or trouble reports.

## APPENDIX -- EMACS Command Summary

^@: sets the mark at the cursor position
^A: moves to the beginning of the line
^B: moves back one character
^C: capitalizes the current character
^D: deletes forward one character
^E: moves to the end of the line
^F: moves forward one character
^G: quits from any command in progress
^H: deletes backward one character
^I: inserts a tab
^J: opens a new line and moves to the beginning of it if the next line is non-empty, otherwise down one line
^K: Kills to end of line (with argument, kills multiple lines)
^L: refreshes the screen
^M: opens a new line and moves to the beginning of it if the next line is non-empty, otherwise down one line
^N: moves down one line
^O: opens up a new line
^P: moves up one line
^Q: quotes the next character
^R: starts a reverse search
^S: starts a search
^T: transposes the next two characters
^U: multiplies the argument by 4
^V: moves to the next page
^W: kills the current region (between cursor and mark)
^X: is a prefix for more single character commands, type character or '*' for all
^Y: restores last killed text (leaves cursor and mark around it)
^Z: exits immediately
^[: Makes the next character a meta character
^^: Causes the last returned result to become the argument
^?: deletes backward one character
M-^L: Re-displays with current line at top of page
M-^M: Mails the current buffer
M-^Q: Returns the next input character (in a macro)
M-^R: Regular expression query replace
M-^S: Regular expression search
M- : sets the mark at the cursor position
M-!: gets and executes a shell command
M-": Auto Fills the whole buffer
M-#: Plays the game of Life
M-$: Executes a command, saving the output in buffer .exec
M-/: starts a comment
M-<: moves to top of file
M->: moves to bottom of file
M-?: explains the next character
M- Converts its argument to a character and inserts it
M-_: Underlines the next word

M-a: Moves to beginning of sentence
M-b: moves back one word
M-c: capitalizes the next word
M-d: deletes the next word
M-e: Moves to End of Sentence
M-f: moves forward one word
M-g: Moves to a specific line (its argument)
M-m: Displays active modes
M-p: Puts the current region in the kill buffer without killing it
M-q: Quotes the next character and adds the 0200 bit
M-r: starts query replace
M-s: Gives EMACS statistics
M-t: Prompts for terminal type
M-v: moves back one page
M-w: Puts a wall chart of explanations in the buffer
M-y: Replaces the last restore() with the next text in the kill buffer.
M-z: kills emacs with a core dump (for debugging)
M-{: Enters a command sequence (in a macro)
M-}: Exits a command sequence (in a macro)

Control-X commands:

^X^B: Changes Buffers (Change to * lists active buffers)
^X^C: exits gracefully (after asking whether or not to save the buffer)
^X^D: Changes the working directory
^X^E: Calls emacs recursively taking input from the terminal
^X^F: Edits a file in its own buffer (if file has been read into a buffer, moves to it)
^X^I: Re-directs input from a file
^X^K: Kills a buffer
^X^L: Loads a file full of macro definitions
^X^M: Sets mode from argument (prompts for mode name) and string if necessary
^X^O: Switches between windows
^X^Q: Returns the character under the cursor (in a macro)
^X^R: reads a new file
^X^S: saves the buffer in the current file (if modified)
^X^T: Prompts for a buffer name and inserts the text between the cursor and the mark into the named buffer.
^X^W: writes a new or old file
^X^X: exchanges the mark and the cursor
^X%: Exchanges the top of the kill stack with another item
^X-: Pops the kill stack
^X1: Exits two window mode
^X2: Enters two window mode
^X<: Pushes a string from the tty or macro text into the kill stack
^X>: Duplicates an item on the kill stack
^XB: Puts the buffer name into the buffer
^XF: Puts the file name into the buffer
^X^: Enters a "while" loop (in a macro)
^Xd: Defines macros from the current buffer

^X| : Begins a conditional execution sequence (in a macro)
^X : Performs arithmetic or logical operations (in a macro)