

A. COMPARATIVE STUDY OF THE USAGE OF THE ED
AND EX FAMILIES OF EDITORS ON THE
UNIX OPERATING SYSTEM

By

SHAKIR MAHMOOD HUSSAIN

Bachelor of Arts

Al-Mustansiriyah University

Baghdad, Iraq

1976

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 1984


Dedicated to my lovely wife, Suaad
and beautiful children,
Raghad and Zaid

Thesis
1984
H97250
Cop. 2

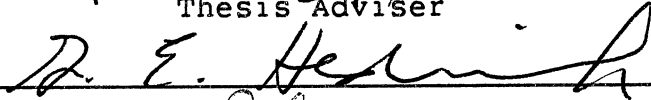


A COMPARATIVE STUDY OF THE USAGE OF THE ED
AND EX FAMILIES OF EDITORS ON THE
UNIX OPERATING SYSTEM


Thesis Approved:



Thesis Adviser







Dean of the Graduate College

PREFACE

This study is concerned with the usage of text editors on the Unix Operating System in the Computing and Information Sciences (CIS) Department at Oklahoma State University (OSU).

I would like to express my gratitude to my major advisor, Dr. Mahir S. Ali, for his valuable advice and guidance provided during this project. Also I wish to thank Dr. G. E. Hedrick and Dr. S. A. Thoreson, my committee members, for their suggestions and Dr. D. W. Grace for his substitution during my oral examination. I would also like to thank Dr. A. N. Walker at Nottingham University, England, for his help.

Deep appreciation is expressed to my wife, Suaad, for her love, understanding, and encouragement through my study.

I want to include a general expression of gratitude to all my friends, students, and professors who have made my years at Oklahoma State University an enjoyable and educational experience.

Lastly, but not least I sincerely appreciate the Iraqi Government for its financial support during my graduate study at OSU.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
Literature Review	3
Intent and Outline	5
II. HISTORY AND TYPE OF EDITORS	7
Historical Development of Editors	7
The Editor: A System Viewpoint	14
Types of Editors	19
III. UNIX EDITORS	27
Line Oriented Editors	27
Full Screen Editors	32
Modification on Editors	33
IV. MONITORING AND ANALYSIS	35
The Monitor Program	35
Data Analysis	36
V. CONCLUSIONS AND RECOMMENDATION	50
Suggested Further Work	51
SELECTED BIBLIOGRAPHY	54
APPENDIX A - Prefix B+_Tree	56
APPENDIX B - Editors Commands	60
APPENDIX C - Usage of Editors Commands	69

LIST OF TABLES

Table	Page
I. OSU/CIS Population	37
II. Unused Commands	38
III. Popular Commands (A)	41
IV. Popular Commands (B)	43
V. Similar Commands in Ed and Ex	45
VI. Similar Commands in Ued and Vi	48
VII. Ed Line Editor Commands	60
VIII. Ex Line Editor Commands	62
IX. Ued Full Screen Editor Commands	64
X. Vi Full Screen Editor Commands	66
XI. Usage of Ed and Ued Editor Commands	69
XII. Usage of Ex and Vi Editor Commands	72

LIST OF FIGURES

Figure		Page
1.	Communication Between Text Editor and Terminal For File Processing	2
2.	The Editor: a system architecture	15
3.	Elements of the Editing Component	18
4.	Unix Editors	28
5.	Organization of B+_Tree	57
6.	B+_Tree: Structure of Root Node	58
7.	B+_Tree: Structure of Index Level Node	59
8.	B+_Tree: Structure of Leaf Level Node	59

CHAPTER I

INTRODUCTION

Editing is the process of creating, examining, and updating text, programs, and data files. The editing process may be viewed as the transformation of an existing string of symbols to a new string of symbols. The editor is a valuable tool for manipulating files in an interactive mode under some commands which are provoked by the user from a terminal.

The editor does not work on the file itself, but rather a copy of the file is made at the beginning of the editing session to prevent loss of data in case of a system failure. Figure 1 shows how the processing of a file is carried out by a text editor from a terminal. In this figure, the text editor makes a copy of the user's file in the main memory from the secondary memory, and all commands are executed on the copy file (17). Modifications that are done during the editing session do not appear on the original file until the new file (copy file) is saved (using write command).

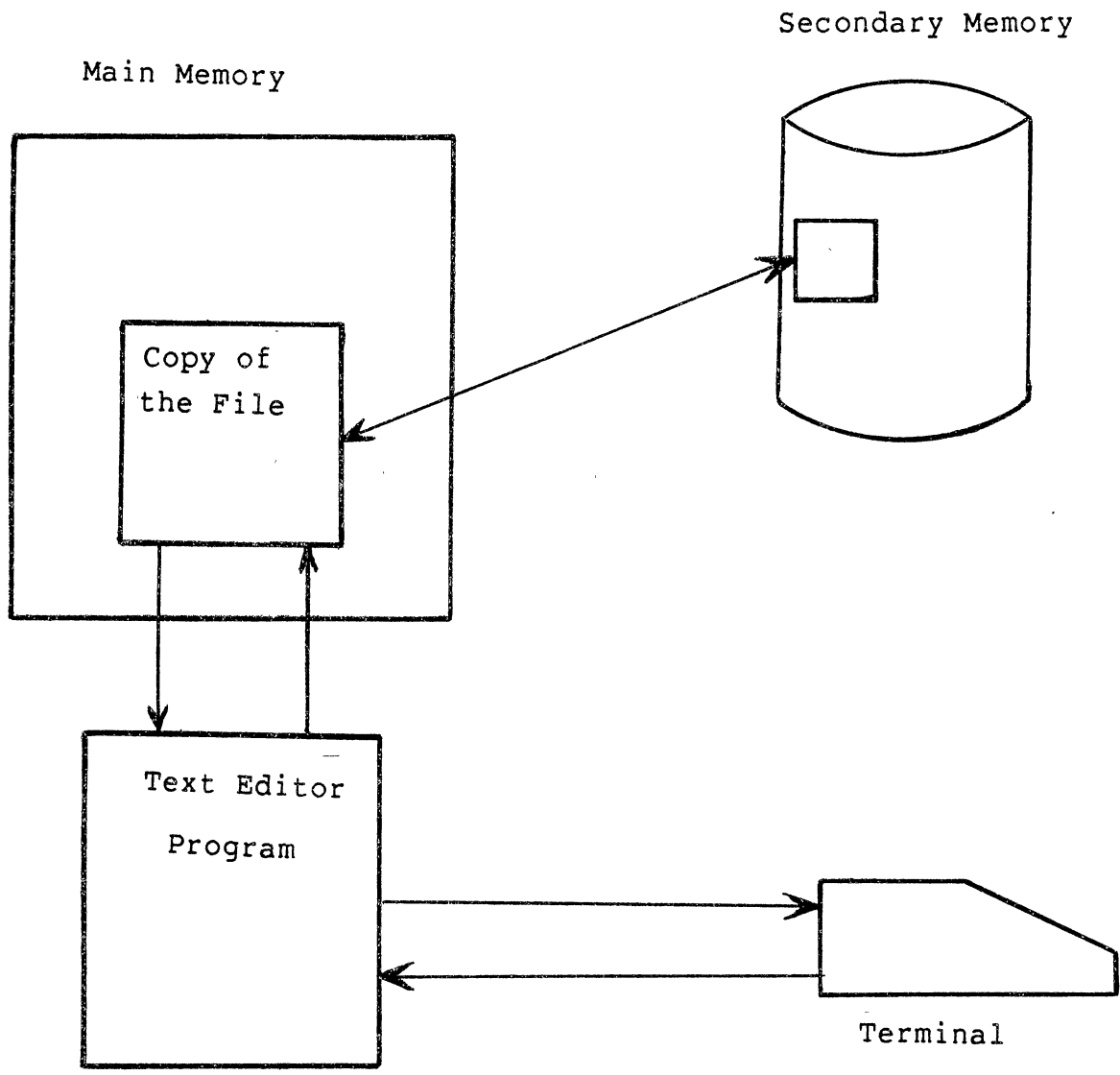


Figure 1. Communication Between Text Editor and a Terminal for File Processing

Literature Review

Fraser (6) defined "editing" as the process of examining and modifying data. Though most editing programs edit text, other types of data need editing too: utilities that delete and rename files edit directories and interactive debuggers edit binary core images. Typically, each utility has its own command language and command scanner. However, each of these utilities is just another form of "editor" and, with careful design, might share the system text editor's command language and scanner.

The general functions of editors, as described by Embly et al. (5) are:

1. Create and modify source programs.
2. Prepare documents.
3. Examine and retrieve portions of a program, text, or a data file.

A text editor is often the primary interface between the user, the system, and the program. An editor must therefore be easy to use and efficient in using computer resources (11).

Stearns (19) classified text editors into two types:

1. Line editor: it has simple form and requires little system programming.
2. A full screen editor: it is more convenient to use, but it requires more system programming and more memory space than line editor. For example, the object code for ed

(the standard line editor developed by Bell Laboratories) is 16776 bytes and ued is 86260 bytes.

The Unix* system as distributed by Bell Laboratories (18) does not have a full screen editor. Instead, it has a line editor, "ed". At Nottingham University, England, the "ed" editor was extended to "ued" to include a full screen mode (22). The University of California, Berkeley (UCB) developed a line editor "ex" which is based on the Bell Laboratories line editor "ed". UCB also developed a full screen editor "vi" which is based on "ex".

The most important characteristic of an editor as described by Deutsch (3) is its convenience for the user. Such convenience requires a simple and mnemonic command language, and a method of text organization which allows the user to think in terms of the structure of his text rather than in some framework fixed by the system.

Embly et al. (5) in 1981, observed from a survey carried out on a 900 of users that the following features characterize a good editor:

1. Self-descriptiveness.
2. User control.
3. Ease of learning.
4. Problem-adequate usability (minimize details the user must know and deal with).
5. Correspondence with user expectations.

*Unix is a trademark of Bell Laboratories

6. Flexibility in task handling.

7. Fault tolerance.

Intent and Outline

The Computing and Information Sciences (CIS) Department at Oklahoma State University (OSU) has a Perkin Elmer 3230 computer running Unix and has line editors ("ed", "ex") and full screen editors ("ued", "vi").

It is proposed that a project to perform a comparison of the use of the facilities in both ("ed", "ex") and ("ued", "vi") is needed to show the similarities/differences between these editors and to have a deep understanding of the "editors at work". The project is to be carried out in two parts:

1. Develop a monitor program to monitor the use of ("ed", "ex") and ("ued", "vi") facilities for a period of time. The criteria for deciding on the "length" of the monitoring period are explained in chapter IV.
2. Develop program to analyze the data collected (see chapter III for the type of data items) from the monitor and to present the information in a useful format.

Historical development of editors is discussed in Chapter II. Also information about the system viewpoint of editors and type of editors is provided in the same chapter.

Chapter III, covers the type of editors available on the Unix Operating System.

Chapter IV is the analysis of the data collected on ("ed", "ex") and ("ued", "vi") editors.

Chapter V gives a summary of this thesis, its conclusions and some suggestions for further work.

CHAPTER II

HISTORY AND TYPE OF EDITORS

The primary reference for this chapter is a paper by Meyrowitz and Van Dam (16). This depends on this reference which is intended as a summary. For further details, the reader is referred to the original paper by Meyrowitz and Van Dam.

This chapter will cover the development of editors, the system viewpoint of editors, and then the types of editors that are available.

Historical Development of Editors

Noninteractive editors were the first editors to be implemented. They began with the manipulation of "unit record" punched cards. The basic unit of information was the 80-column line; the user made corrections on a line-by-line basis, retyping mistyped cards. The card gave the programmer new freedom compared to toggling in bits at the system console. The user could store information in readable form, and then access this information, changing its order, discovering and correcting errors.

Punched card decks had many disadvantages, such as the rearrangement of the entire box of cards when the box was

accidentally dropped. More seriously, editing a small part of a large document required feeding the entire document. Correction of small errors, such as single-character errors or double-character transpositions, required retyping the error and replicating the other characters with the duplication facilities of the keypunch. Replacing a word with a word of different size required duplicating all the characters prior to the word and retyping all the remaining characters from the new word to the end. If the incorrect card was almost completely filled with characters, inserting a new word might cause an overflow in the contents; therefore insertion of one or more new cards required handling the overflow. Global change was much more difficult, because it required finding all occurrences of the pattern manually and then replacing the new pattern again manually; if the new pattern were larger than the old pattern, multiple overflows could happen easily.

In 1960s the use of cards was very common. A batch editor created to remove the problems of dropped cards and retyping, and in some versions provided new operation such as global replacement of a pattern. The main idea of batch editor was to store the programmer's initial deck of cards as a card-image tape or disc file. Each card was referenced by a unique sequence number. Changes were made by creating an edit deck composed of cards containing editing requests, and running the deck through the batch editor program. For example, the request "in card 107, correctly spell the word

'data'" would be made by typing the sequence number 107 on one card followed by a card containing the new contents of line 107, or more simply by using one card contain sequence number and editing command as in

```
107 REPLACE/DATE/DATA/
```

Batch editors solved problems appeared when using cards; but there were several disadvantages. Programmers needed to have a line-printer listing of the entire deck cards before making any change. Also because batch editors relied on sequential storage media such as magnetic tape, the user could only step through card images linearly, stopping at lines which needed correction, and making correction according to the editing command. To go backward the file need to rewind and start again.

Line editors were implemented in systems like IBM's MTST (16), which used a selectric typewriter as an input device and small magnetic tapes and/or cards as storage media. The utility of these initial line editors was limited by the typewriters, which supported the viewing of only one line at a time and had very slow printing speeds. Also problems appeared with updating when the user required going forward and backward to get the desired location on the file.

In the mid 1960s, interactive line editors were designed to allow the user to create and modify disc files from terminals. These editors attached either fixed or varying (sequential relative to the top of the file) line

numbers to lines of limited length (initially 80 characters), allowing the user to reference any part of the information. Examples of these include ATS and VIPcom (16). Simple commands languages allowed the user to make corrections within a line or even within a group of contiguous lines, using almost the same syntax as used in batch editors.

Another advance was the creation of the context-driven line editor, which allowed the user to identify the line containing the target of an operation by specifying a character context pattern for the editor to match, rather than by giving an explicit line number. An example of the context-driven line editors was the editor running on the IBM 7090. At this point in the history of editing, users were still forced to think about multiline entities, such as paragraph and program blocks, as groups of integral lines, usually in card image format; no interline commands were available that would, for example, delete spanning from the middle of one line to the middle of the next line.

The first break from the 80-column card image came in the form of variable-length line editors, specified by (com-Share's Quick Editors "QED") (3). The line was still the main element of operation, but now each line could be of "arbitrary" length. Initially, these lines were actually limited to some maximum. QED was the first that used the notion of "superline" (limited to 500 characters in length), which the on-line display process broke into viewable lines

of 80 characters each until the superline was exhausted. Later a variable-length line editors were designed and implemented. By removing the card image orientation of the editor, the variable-length line editor had strong and beneficial impact on the versatility of text processing. Another far-reaching result of the invention of variable-length line editors was that displayed text was no longer considered to be a one-to-one mapping of the internal representation, but rather a tailored, more abstract view of the editable elements.

Even with superline editors, three basic problems in manuscript editing remained:

- a. Truncation when the line length was exceeded.
- b. Inability to edit a string crossing line boundaries.
- c. Inability to search for a pattern crossing line boundaries.

The stream editor concept solved all three problems by eliminating line boundaries altogether: the entire text was considered a single stream or string that was broken into stream lines by display routines. An arbitrary string between any two characters could be defined for searching and editing.

Another way of dealing with the limitation of line and superline editors was to use the power of multiline display screen which provided cursor addressability and possibly local buffers, to create what are now called synonymously full-screen, display, or cursor editors. These editors work

either with stream or variable-length lines, offering the user an entire screenful of text to view and edit without regard to line boundaries. An early example of a timeshared display editor is Stanford University; TVEDIT (16).

Commands, represented by control character sequences, could be interspersed with the input of normal text. Users were able to move the cursor to point to the text they wish to manipulate rather than having to describe text arguments in some awkward syntax. Characters could be replaced by simply typing over them. Characters could be deleted by placing the cursor on the character and pressing the delete control character; characters to the right of the cursor moved left so that the cursor seemed to "swallow" characters.

Similarly, for insertion, the characters to the right of the pointer moved to right, reserving a place for the new characters.

A major new way of thinking about editing was introduced as early as 1959 by Douglas Engelbart at Stanford Research Institute (16). His NLS (oNLine System), implemented in the 1960s to create an environment for on-line thinking and authoring, showed the power of display terminals, multicontext viewing, flexible file viewing, and a consistent user interface. NLS was the first structure editor in that it provided support for text structure and hierarchy, not just for manipulating raw string of text: the user could manipulate documents in terms of their structures, not only their content (16).

Hansen's EMILY (7) extended the concept of the structure editor and developed the syntax-directed editor, in which the structure imposed on a program being edited was the structure of the programming language itself. Users were able to manipulate logical constructs, such as do-while loops and their nested contents, as single units.

In the late 1960s, general-purpose time-sharing facilities typically supported only simple interactive line-editing and batch-formatting facilities for line-printer output. These facilities were barely adequate to create and modify programs and rudimentary documentation. By the early 1970s, text processing had become sufficiently important to be the single dedicated application on both stand-alone and timeshared minicomputers. Since these minicomputers did not need to support general-purpose computing facilities, manufacturers were able to offer comprehensive editing capabilities as well as features oriented toward document production such as database management, information retrieval, work-flow management that were usually unavailable on general-purpose system. For a time, owners of these systems often had more text-processing power than those with much more expensive and much larger general-purpose computers. Examples of dedicated word processing system include CDT, Lainer, DEC-Word/11, and NBI (16).

An important milestone in text editing and text processing was the early 1970s development and mid 1970s

acceptance of the Unix timesharing system (18), the first general-purpose computing environment in which text utilities were given as much weight as programming utilities. In Unix, a suite of utilities (the ed text editor, the troff and nroff text formatter, the tbl table formatter, and the eqn equation formatter,) (11) introduced and popularized an extensive set of text tool in the general-purpose computing community.

Current research in the editing field is focused upon several overlapping areas. One is that of providing a consistent, editor-based interface throughout a computer system (6). This allows many common functions, such as renaming files, searching through directories, and debugging programs, to be performed as editing operations. For example, to rename a file, one would type over the old file name in a listing of available files that would appear on the screen; in debugging a program, one would be able to edit the values of displayed variables. Other research topics include generalized structure editors, powerful syntax-directed editors with program-tracing capability, and interactive editor/formatters.

The Editor: A System Viewpoint

The architecture of any editor can be represented by (Figure 2). This general form can be implemented regardless of the particular computers and the features can be found in that particular editors (16).

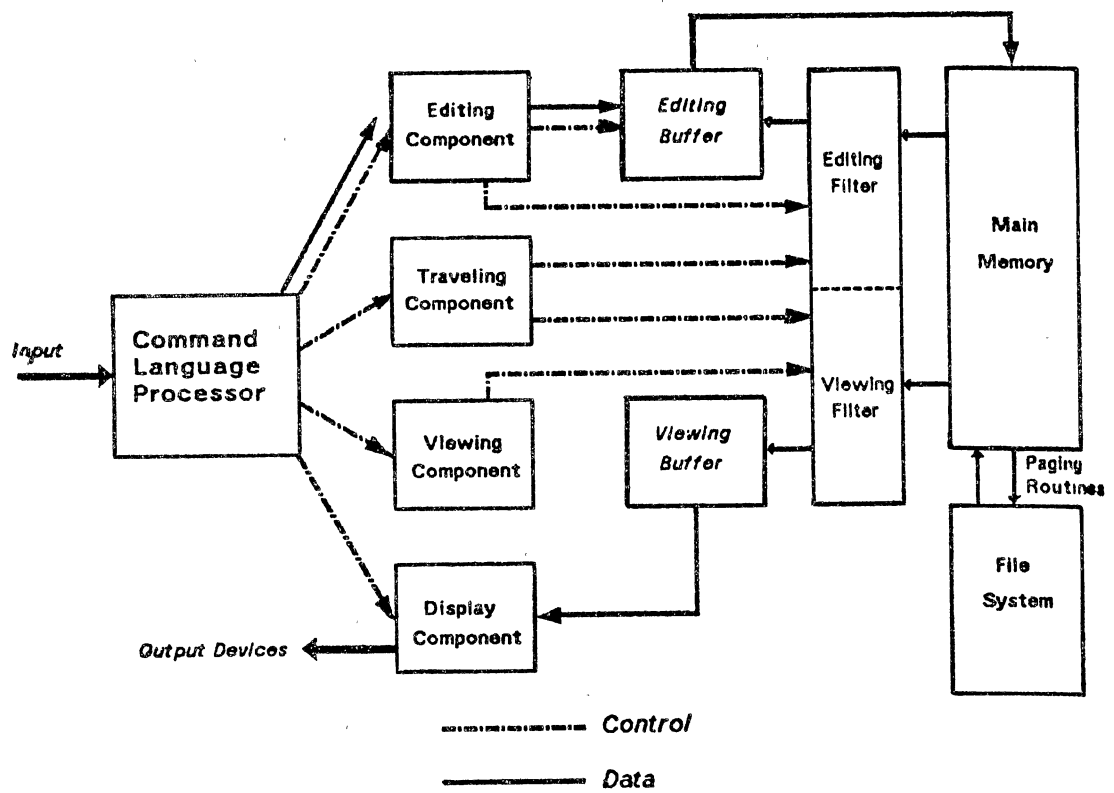


Figure 2. The Editor: A System Architecture

The command language processor accepts input from the user input device, then there are two levels of process to be done. Lexically analyzes and tokenizes the input stream is the first level and the second one is syntactically analyzes the accumulated stream of tokens, then after finding the legal composition of tokens the appropriate semantic routines will be invoked.

At the syntactic level, the command language processor may generate an intermediate representation of the proper editing operations instead of explicitly invoking the semantic routines. This intermediate representation is decoded by interpreter that invokes the suitable semantic routines. The semantic routines of the editing component then operate on the editing buffer, which essentially a filtered subset of the document data structure.

Viewing a document is similar to editing a document. The part of the data to be viewed is determined by the current viewing pointer maintained by the viewing component of the editor. The current viewing pointer can be set or reset explicitly by the user or implicitly by the system as a side effect of the previous editing operation. When the display needs to be updated, the viewing component invokes the viewing filter. The viewing filter filters the documents to generate a new viewing buffer based on the current viewing pointer as well as viewing filter parameters. These parameters are specified by the system and the user. The

viewing buffer may contain the current line or the null string in line editors, while in a full screen editors it may contain a rectangular cutout of the quarter plane of text. This viewing buffer is then passed to the display component of the editor, which maps it to a window (viewing buffer) or viewport, a rectangular subset of the screen, to produce a display.

The editing and viewing buffers, while independent, can be related in many ways. In the simple case they are identical (case of full screen editors, in which the user edits the text directly in view on the screen instead of specifying material with typed commands), see Figure 3 (16).

The editing and viewing buffer can also be disjoint, for example, in the University of California/ Berkeley Unix editor "ex" (20), a user might travel to line 100, and after viewing it, he decide to change all occurrences of "line editor" to "editors" in lines 10 through 60 of the file by using the substitute command:

```
10,60s/line editor/editors/g
```

As a part of this editing command, there is implicit travel to the line 10 of the file. Lines 10 through 60 are filtered from the document to become the editing buffer, and successive substitutions take place in the editing buffer without corresponding updates of the view. If the pattern is found, the current pointers are moved to the last line that the pattern is matched, and that line becomes the default contents of both the editing and viewing buffers.

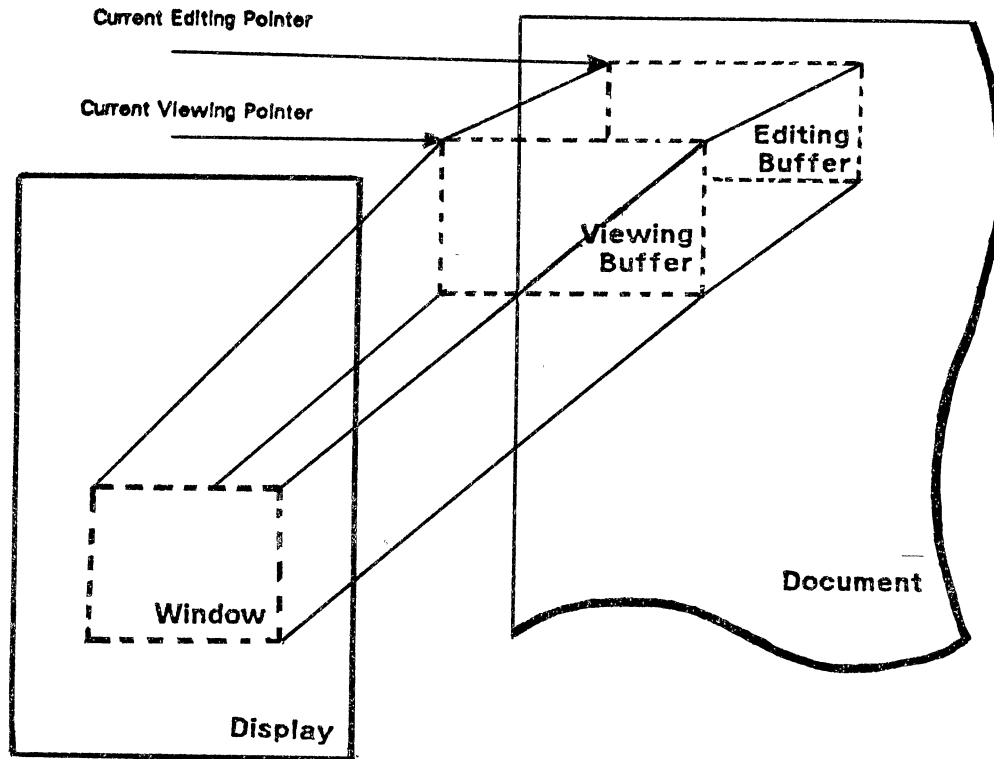


Figure 3. Elements of the Editing Component

When the pattern is not found the default editing and viewing remain on line 100.

Types of Editors

Several types of editors have been implemented:

Line-oriented Editors

The concept of line editors is that of editing virtual card images; the line editor constantly visits the limitations of this outdated representation of data on the user. Drawbacks of this type of editor are pattern searches and edits that do not cross line boundaries, and overflow and subsequent truncation of fixed-length lines. Examples of line-oriented editors are:

1. IBM's CMS

The IBM's CMS editor (16) which is a classic example of fixed-length line oriented editor with a textual interface, designed for a time sharing system in which terminals lack cursor motion keys and function keys.

2. SOS

SOS (4) is another example, like CMS editor, is a line editor designed for editing on time sharing system, specifically a wide range of Digital Equipment Corporation (DEC) computers.

3. Unix ed

The Unix text editor, ed (11), is another type of line-oriented editors similar to CMS editor and SOS, but it has a variable-length line property.

Stream Files

Stream editors act upon a document as a single, continuous chain of characters, as if the entire document were a single, indefinitely long character string, rather than act upon fixed-length or variable-length lines. By doing so, they avoid line editor problems such as truncation and inability to perform interline searching or editing. An example of this type is TECO (16).

TECO, the Text Editor and CORrection, which is an interpreter for a string processing language. TECO can be used interactively as a stream-oriented editor; its basic commands can also be used as building blocks to provide quite elaborate editing operations. Many variations exist (DEC TECO and TENEX TECO); with varying capabilities and syntax. The conceptual model considers a document to be a sequence of characters, possibly broken into variable-length virtual pages by formfeed characters, and into virtual lines by line-end characters. Pages may be combined in an in-core editing buffer considered to be simply a varying-length string whose length may grow up to the in-core memory available.

Display Editors

This category includes several editors based on work done by Deutsch (3) and on the work of Djourp and Irons (8), as well as several editors with an Irons-like model. In the Irons conceptual model, text is conceived of as a quarter-plane extending indefinitely in width and length, with the topmost, leftmost character the origin of the file. The user travels through this plane by using cursor keys and changes characters by overtyping. At any time, the user sees an accurate portrayal of the portion of the file displayed. Text is input on the screen at the position of the cursor. The environment is "modeless"; since all typing on the screen is considered text, commands must be entered either through function keys, control characters, and escape sequences, or by moving the cursor to and typing in a special command line at the bottom of the screen. Examples of display editors are:

1. Brown's bb

Brown's bb (16) is a typical example of the Irons model editor, running under the Unix operating system on VAX11/780. It makes use of a wide range of function keys for interaction. One of the bb's extensions of the model is the maintenance of an up-to-date temporary file on disc along with a linked list of changes that have been made to the old file. This change history serves as the backbone of the undo

command, which is capable of reverting changes back to the beginning of the editing session.

2. Yale's Z editor

Yale's Z editor (16) extends the general Irons functionality by providing facilities that aid in program creation while maintaining the general-purpose functionality of the editor.

Editor commands are entered using control characters coupled with the cursor keys. Function keys are not used; the developer disliked the fact that the user's hand must be moved from the typewriter keyboard to use them. Software allows overloading of the standard ASCII character set by using certain keys as shift keys. The interaction language also supports the overloading of each editor command.

Graphics-based Interactive Editor/Formatters

Examples of this type are:

1. Xerox PARC's Bravo

Xerox PARC's Bravo (16) is one of the first of the interactive editor/formatters based on the display of high-resolution, proportionally spaced text. Bravo allows the creation and revision of a document containing soft-typeset text with justification performed instantly by the system. The conceptual model is of a continuous scroll of typeset text that can be paginated when desired.

2. ETUDE

ETUDE (16) is a document production system designed to extend the functionality of conventional word processing systems while reducing the complexity of the user interface.

General-purpose Structure Editors

Structure editing, pioneered by Englebart with NLS (16), has been "rediscovered" as an alternative to standard character-oriented methods of editing. Since most target applications have some innate structure (e.g. manuscripts are composed of chapters, sections, paragraphs), the philosophy of structure editors is to exploit this "natural" ordering to simplify editing. The most common representation is a hierarchy of elements. Examples of general-purpose structure editors are:

1. NLS/AUGMENT

NLS was a product of research at Stanford Research Institute (now renamed SRI, international) (16) between the early 1960s and late 1970s. Renamed AUGMENT and marketed by Tymshare, Inc., NLS is one of the seminal efforts in the field of text editing and office automation; indeed, many of its features are being reexamined and reimplemented today.

2. Burkhart/Nievergelt Structure Editor

Burkhart and Nievergelt at the Institute for Information in Zurich have designed a family of structure-oriented editors called XS-1 (16). The designers contend

that the basic sets of editing operations, regardless of the target being manipulated, are similar, and that "a universal structure defined on all data within a system" exploits that similarity to its greatest advantage.

3. Fraser's s

Fraser's (6) is an attempt to provide standard editing primitives that can be used to build a variety of editors. "s" allows the programmer quickly to create different front ends for a text editor so that various targets can be modified using existing editing routines. The philosophy behind s is that many computer utilities are simply editors in that they accept a particular input syntax and modify the existing representation and/or state of their particular data. Rather than producing languages and scanners for each application, s attempts to use a generalized structure and a generalized text editor nucleus for editing all application.

Syntax-Directed Editors

Syntax-directed editors attempt to increase the productivity of the programmer by removing the time-consuming process of eliminating syntax errors. Syntax editors are structure editors that ensure that the structure always is constrained to preserve syntactical integrity. Often syntax-directed editors do not merely recognize the syntax and translate the user's actions into linear text, but instead parse the input into an intermediate form that can be used to generate code. Here the editor is both a

tool for the programmer and a tool for the compiler/interpreter. An examples of this type are:

1. Hansen's EMILY

Hansen's EMILY (7) is one of the earliest syntax-directed editors. Rather than typing in arbitrary text, the user creates and modifies text by graphically selecting units of text (template) that are constructs in a programming language. Text is created with a sequence of selection. The screen is divided into three areas: text, menu, and message.

2. Fraser's sds

Fraser's sds is a general structure editor driven by a grammar that describes a hierarchical data structure. The user-viewable part of sds is a screen editor with displays a current record of some tree structure. The cursor keys down, up, left, right, and home allow the user to move down to a node field, back up, left or right to adjacent fields, or to the root of the structure.

Word Processors

Examples of Word Processor are:

1. WordStar

WordStar (16) is one of the most popular word processing programs available for home computer system. It runs on a variety of systems under the CP/M operating system, using the CP/M file system to maintain its files.

2. NBI System 3000

The NBI System 3000 is another popular commercial word processing system. It has a stand-alone processor, with file storage on floppy disc. Its conceptual model is very similar to that WordStar described earlier (16).

Integrated Environments

RIG and Apollo are examples of this type. RIG and Apollo systems are based on the concept of a display or window manager as the primary interface to the system. These display managers give the user the ability to create windows on the display surface, move these windows around, and change their size. On the Apollo these windows can overlap; in RIG the windows do not overlap but simply partition the display screen.

CHAPTER III

UNIX EDITORS

The Unix System at the OSU/CIS Department has both line editors and full screen editors. Line editors are "ed", "ex", and "sed". Full screen editors are "ued" and "vi" (see Figure 4).

Line Oriented Editors

Line editors are divided into two types: Interactive editor and Noninteractive editor.

Interactive Editors

1. Ed Text Editor

Ed is the standard line editor on the Unix system. Since "ed" is a line editor, any operation to be performed must specify line or lines on which the operation is to be carried out.

Lines can be accessed in several ways, the most easily understood method of addressing lines is by line number. Other methods for accessing lines are by using the line's textual contents or position of line in the text (end of file, for example).

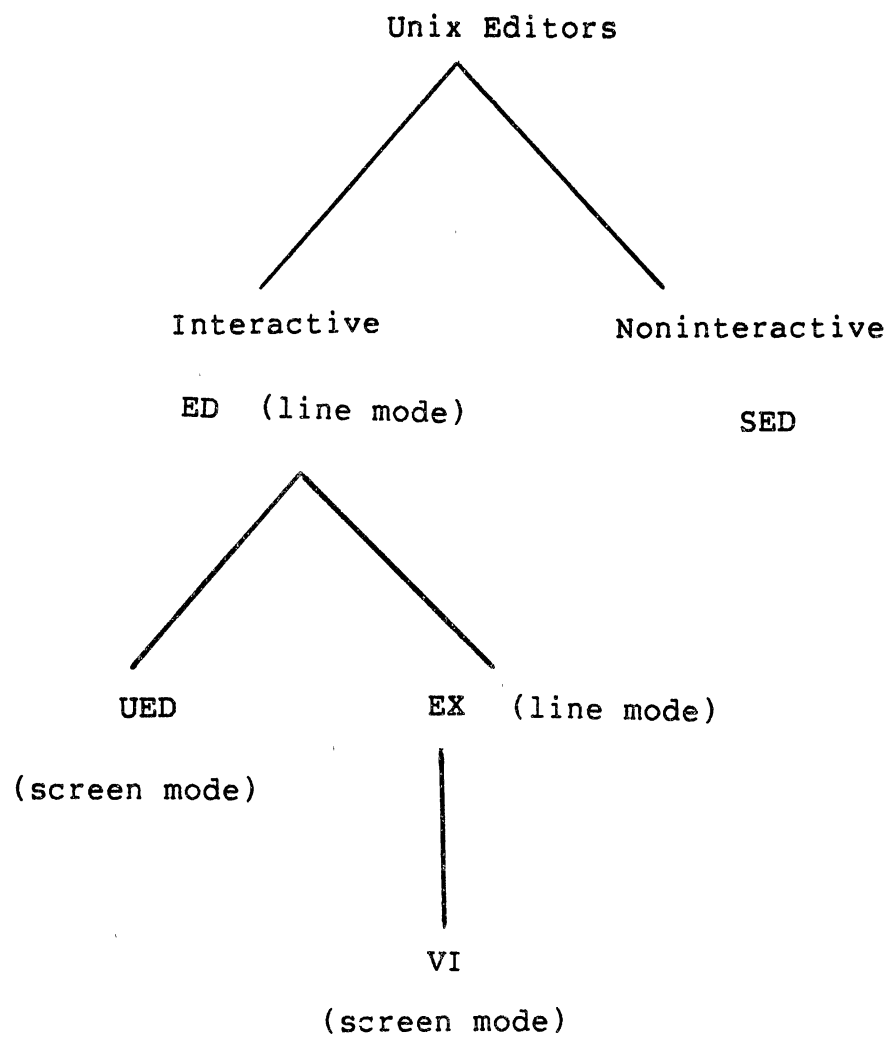


Figure 4. Unix Editors

The format of "ed" commands is:

```
[line[,line]]operation[parameter]
```

The "ed" command consists of an optional line-address, or two optional line-addresses separated by a comma, then a single letter indicating the operation, followed by some other (option) parameter. The form of the parameter is varied for each operation. For example, the move "m" operation, the parameter is the line that the addressed lines are to be moved to; while in the read "r" operation, the parameter is the file name that is to be read. The substitute command is an exception to this rule because between the operation and the optional parameter, the string to be substituted and the new string need to be specified.

Detailed description of the "ed" commands can be found in the Unix System Manuals (20), "A Tutorial Introduction to the Unix Text Editor" (9), or "Advanced Editing on Unix" (10).

2. Ex Text Editor

The "ex" text editor was developed by William Joy of the University of California at Berkeley (23). The "ex" text editor is based on "ed" and therefore users who have experience with "ed" can easily learn and use "ex".

Ex is somewhat easier to use than ed. This is mostly because "ex" is more communicative. It tries to inform the user of the error while "ed" responds by "?" to all types of errors.

Ex has many extensions and improvements to the "ed" editor, as described by McGilton (15), these are:

1. Operations are not restricted to a single character, so they can be remembered more easily - for example, "co" for copy instead of "t" for transpose. However the single-letter operations used in "ed" are retained, so the experienced "ed" user will not have to remember the new ones.
2. Introduction of additional operations not found in ed.
3. Variants of some editor operations, which modify the way in which those operations are performed under certain conditions. Operation variants are invoked by placing a ! character following the normal operation, for example a!.
4. Improved messages for error conditions, with instructions as to how to override the error condition.
4. Editor "options" which modify the overall behavior of ex.
5. Provision of a means of recovery if the system crashes during an editing session.
6. Introduction of a "visual" mode which turns the editor into a screen editor. In this mode, ex is identical with the vi full screen editor. There is also an "open" mode, which provides intraline editing.
7. ex and vi editors react to different terminal types, this is necessary because of the screen editing capability.
(P. 236)

For more details on the "ex" editor commands see Ex Reference Manual (23).

Noninteractive Editor

The stream editor, "sed" (14), is the only noninteractive editor available on the Unix System. Sed, unlike the line or full screen editors works on the original file, instead of a copy of the file. It also edits according to a script of commands stored in a file on the system rather than interactively from a terminal. Sed is designed to be specially useful to edit files which are too large for comfortable interactive editing, to perform multiple "global" editing functions efficiently in one pass through the input, and finally to edit any size file when the sequence of editing commands is too complicated to be comfortably typed in an interactive mode.

The general format of the commands is the same as "ed" commands:

```
[line[,line]]operation[parameter]
```

However there are some important differences, given by McGilton (15), from "ed" :

1. The only operation which can take the optional final parameter is the s (substitute) operation.
2. If no line numbers are specified, the operation is performed on all lines. This is quite different from ed, where the default line is usually dot.
3. Lines can be addressed by number, or by text pattern using fixed character strings, or by regular expressions. Because the default mode of operation is global, there is no concept of "current line", nor of relative line address. Line numbers are absolute in the file.

4. Operations that require text input (a, i, and c) have a different format from the same operation in ed.
5. Many ed operations have no counterpart in sed. In particular, the m (move) and t (transpose, copy) operations do not exist.
6. Contrariwise, there are some operations that are available in sed that do not exist in ed. One of these is y (for transform, or maybe transLYterate - not too mnemonic).
(P. 220)

Full Screen Editors

1. Ued Editor

ued is an extended version of the standard Bell line editor "ed", developed by Dr. A. N. Walker at Nottingham University, England (22). Enhancement have been added such as a full screen facilities, much better pattern searching in the line mode, and a better interface with the Unix Shell (20).

Modification on the file can be performed by displaying a portion of the file to be modified on a terminal screen called "window". Within that portion the cursor can be moved around to the position where the modification is needed to be performed.

All "ued" commands in full screen mode are control characters, and act on the cursor's current screen position. Details on "ued" commands can be found in "Unix Program's Manual" (20).

2. Vi Editor

Vi is another editor that is associated very closely with the "ex" text editor (20), but it is classified as a full screen editor. This editor is called a "visual" editor, "screen" editor, or a "disply" editor. In "vi", the portion of the file to be modified is displayed on the terminal screen. This is often named the window. Within this window the cursor can be moved around to control where changes are to be made, and then the changes can be made by replacing, adding, or deleting text. The window can be moved up and down to display any portion of the file, therefore making it possible to access any section of the file.

Vi has no default mode, like "ued" where the default mode is "write". Any modification (insertion, deletion, or replacement) needs an explicit command in "vi".

It is possible to change mode from "vi" to "ex" (for example, full screen mode to line mode) and vice versa. The reason for this is that "ex" and "vi" are linked together to the same code.

Vi has a wide range of commands, a detailed description of these commands can be found in "An Introduction to Display Editing with vi" by William Joy (24).

Modification on Unix Editors

For the monitoring period, counters have been added to the Unix editors: "ed", "ued", and "ex" and "vi". The

purpose of these counters is to keep track of how many times each command is invoked. Permanent counters are stored in two files; one for "ed" and "ued", and the other is used for "ex" and "vi" commands. Temporary counters are used for each editing session and a counter is incremented by one every time its related command is invoked. When the editing session is finished the permanent counters are modified using the values in the temporary counters.

Finally, chapter IV describes the analysis of the total accumulative values (i.e, at the end of the monitoring period) of these counters.

CHAPTER IV

MONITORING AND ANALYSIS

This chapter covers the comparison of the usage of all commands for both line editors (ed, ex) and full screen editors (ued, vi). The empirical usage of the commands of all the editors was continuously monitored from March 12, 1984 to April 30, 1984.

The Monitor Program

The purpose of the monitor program is to maintain counters for all commands of each editor for every user.

Prefix B+_tree has been used to implement the monitor program. A full description of the structure of the monitor program and its implementation is given in appendix A.

The monitor program was invoked during the second half of the spring semester, 1984 at the CIS Department, OSU.

Two main factors have been decisive in determining the length of this period, these are:

1. There are two main semesters in one academic year, the fall and the spring semesters. The majority of the graduate students (the only group of students allowed to use Unix) join the CIS Department in the fall semester and are likely to be unfamiliar with the Unix system and

its editors. By the beginning of the spring semester, most of all graduate students who are going to use the system have already started or are beginning to start using the system. By the end of the first half of the spring semester, all the students should have learned to use the system, including the editors, well enough not to be called "beginners". It is also expected that by the beginning of the second half of the spring semester, most of the work which is being carried out on the system is in connection with major projects related to the graduate level courses. It is our believe that the selected period (the second half of the spring semester) represents the heaviest workload the system has to deal with.

2. We have decided to run the monitor continuously (24 hours a day, seven days a week) simply because the system is used all the time and some users prefer, or for personal reasons, use the system late at night or very early in the morning or mainly during the weekends. This made it difficult to decide upon an unbiased "sampling period" during the day or/and the night. It was, therefore, felt that the monitor should run for the specified period, which represents 25% to 33% of the annual system usage.

Data Analysis

It was found, at the end of the monitoring session, that 66 users have accessed the editors during the second

half of the spring semester. However, 51 users have used "ed" and "ued" and 44 used "ex" and "vi" while 29 users used both groups of editors. Table I shows the usage of the editors by the CIS users.

TABLE I
OSU/CIS Population

Editors used	Number of users
ed & ued	51
ex & vi	44
both	29
neither	70
Total	136

Appendix B has four tables. Each editor has a table of its commands sorted alphabetically by its commands, also a short description of each command is given in these tables.

Appendix C has two tables one for "ed and ued" editors and the other is for "ex and vi" editors. These tables give the data collected during the monitoring period.

Unused Commands

Table II shows the unused commands during the monitoring period. It was found that only one command in "ed" is not used which is the mark "k" operation. This operation was also not used in "ex". The mark "k" command is used with operations which need to remember line number such as move "m" and copy "t". It is possible to perform these operations without using the mark command by using the actual line numbers.

TABLE II

UNUSED COMMANDS

Editors	Commands	Description
ed and ued	k	mark line
ex and vi	k	mark line
	<	shift line one tab to left
		print next line but one
	se	set terminal type
	Ctrl z	suspend editor's session and temporarily return to shell
	==	indent for LISP

Also in "ex" editor, it is possible to change line mode to full screen mode by using the command "vi" and the mark operation is available in it under the name "m" and it was used by the users (table XII, Appendix C); therefore, the mark command "k" is not needed unless the user works with line mode only.

The command "<" (shift line one tab to left) is an "ex" command and it was not used because this operation can be performed within the full screen mode by using the "<<" command which does the same operation as "<" commands does within the line mode.

The command "se" (set terminal type) also is one of the unused command in "ex". This command is not used because the terminals that are available at the CIS Department are set by default in the shell script.

Indentation for LISP command "==" is not used simply because LISP is not available on the system.

The reasons which can be thought of for not using the command "Ctrl z" (suspend editor's session and temporarily return to shell) is that most users use the command "!" which allows users to exit temporarily from the editor and perform one unix command. The other reason which can be thought of is that users seem to prefer to exit permanently from the editor if they wish to execute more than one unix command and then return to "vi".

The last command which was not used in ex is "|" (print next line but one). The only reason which can be thought of for not using this command is that it is not useful for the applications of the CIS population.

Popular Commands

It was found that only seven commands of "ed and ued" editors make 83.92% of the total commands used (table III), and five commands of "ex and vi" editors make 80.49% of the total commands used.

From table III, the top three commands used in both editors ("ed and ued" and "ex and vi") perform the same operations. The "->" and "l" commands scan the line from left to right one character at a time. Users need to scan lines to position the location for making any correction (deleting, replacing and adding new characters). Also scanning from right to left ("<-" and "h") is needed for the same reason that is given above.

There is a need for deleting a character(s) from a line without leaving a blank in the deleted location. This can be done by using "Ctrl R" command in "ued" and by using "x" in "vi"; therefore, these commands are used 4.6 and 8.5 percent of the total commands used respectively.

In the "ued" editor, the command "Ctrl W" is used to widen the line to the right from the location of the cursor. It is used to insert a space in a line. This operation can

TABLE III
 POPULAR COMMANDS (A)
 USAGE OF COMMANDS
 GREATER THAN 3%
 OF THE TOTAL
 COMMANDS

ed and ued		ex and vi	
command	% of total	command	% of total
* ->	19.69	l	35.47
* <-	16.66	h	29.16
* Ctrl R	4.81	x (vi)	8.50
↑	13.00	Ctrl D	3.92
↓	14.65	C	3.44
"RETURN"	10.54	Total	80.49
Ctrl W	4.57		
Total	83.92		

* indicates that the two commands on this line perform the same function.

also be performed with two commands "Ctrl V Ctrl E" which are considered to be one command, but this operation is not described in the "ued" online manual; therefore, the "Ctrl W" is used 4.57 percent of the total commands used.

However, in "vi" editor there is more than one command to insert and widen at the same time such as "a", "I" and "i". This explains why "Ctrl W" is used heavily in "ued".

In "ued" editor, overwriting is allowed by typing over the character(s) directly, while in "vi" any changing must be done through the change command "C". For this reason the change command is used heavily (3.44 percent).

Table IV shows the commands that are used greater than one percent and less than three percent of the total commands used. The "u" command is used to change the editor mode from line mode "ed" to full screen mode "ued". This operation also available in "ex" editor. "vi" and "u" commands used almost within the same percentage of the total commands used. Also the second and the third commands in both editors ("Ctrl C" , "Ctrl D" in "ued" and "Ctrl \ " , "dd" in "vi") perform the same operations as it is shown in table IV.

The command "x" is an "ex" command used to write and quit from the editing session. In "vi" the same operation can be performed by using "ZZ" command. These commands appear in table IV almost within the same percentage while the write "w" and quit "q" commands do not appear in the table under the "ex and vi" editors. The write and quit

TABLE IV
 POPULAR COMMANDS (B)
 USAGE OF COMMANDS
 GREATER THAN 1%
 AND LESS THAN 3%
 OF THE TOTAL
 COMMANDS

ed and ued		ex and vi	
command	% of total	command	% of total
* u	1.28	vi	1.91
* Ctrl C	1.29	Ctrl \	1.24
* Ctrl D	2.12	dd	2.98
Ctrl G	1.59	x (ex)	1.21
Ctrl V	1.28	O o	1.40
TAB	2.20	ZZ	1.22
q	1.08	cc	2.86
r	1.01	Total	12.82
Total	11.85		

* indicates that the two commands on this line perform the same function.

(one command) does not have an equivalent in "ed" and "ued" editors. The quit "q" command appears in table IV with 1.08 percent while the write "w" commands does not appear in the table but it is used with 0.97 percent of the total commands used and the total percentage of "w" and "q" make 2.05 percent which is close to the total of "x" and "ZZ" commands (2.43%). Since the "w" and "q" commands must be used at least once per session, therefore, their usage is heavy as the data collected shows in table IV.

Table III and IV show that a total of fifteen "ed and ued" commands make 95.77% of the total commands used and the remaining commands make the other 4.23 percent. In "ex and vi", only twelve commands make 93.31% of the total commands used and the others make only 6.69 percent. Also from these tables, it is found that there are six similar commands which share the heavy usage in both editors.

Similar Commands in Line Mode

The line editors "ed" and "ex" have some similar commands. Table V shows the average usage of these commands per user during the monitoring period. It was found that most of the "ed" commands have averages higher than the "ex" commands. The reasons for this are, first because of the larger number of commands in the "vi" editor that are available. The second reason is the availability of commands in the full screen mode "vi" which are not available in the full screen mode "ued", but are available

TABLE V
SIMILAR COMMANDS IN ED AND EX

command	ed	ex
	average * per user	average * per user
a	20	1
i	12	1
r	365	5
c	9	1
d	86	9
e	1	4
f	1	3
g	9	1
j	1	2
k	0	0
l	1	1
p	74	16
m	12	1
t	7	1
q	389	31
s	77	11
u(ed) vi(ex)	463	281

TABLE V (Continued)

w	351	83
(" %) (z)	1	1
!	18	4
Number of users	51	44

* Averages are rounded to the nearest digit

in the line mode "ed". These two reasons force the "ued" users to use more line mode commands than "vi" users.

The "a" and "i" commands are used to insert text (one line or more). This operation can be performed in one command in "vi" while in "ued", more than one command is needed to perform the same operation. Users prefer to use one command to do what they need; therefore, the "a" and "i" commands in "ed" editor are used more than in "ex" editor.

The quit command "q" is used more in "ed" than in "ex" simply because there is more than one command available in "ex" and "vi" to perform this function ("x" in "ex" and "ZZ" in "vi"). Also the same reason can be given for the write command "w". Another reason which can be thought of for making the averages of most "ed" commands higher than the "ex" commands is that "ed" is not linked with the full screen editor "ued". "Ed" and "ued" are two different programs while "ex" and "vi" are the same program. From "ex", accessing "vi" commands is possible while in "ed", there is no way to get the full screen mode "ued". However, the line editor commands are available in "ued" mode, except the undo command "u" because this letter is used to get the full screen mode.

Similar Commands in Full Screen Mode

Table VI shows the usage of the similar commands in the full screen editors "ued" and "vi". It was found that most of the operations which need more than one command, such as

TABLE VI
SIMILAR COMMANDS IN UED AND VI

ued		vi	
command	average * per user	command	average * per user
Ctrl q	1	Ctrl L	24
Ctrl V ↓	198	Ctrl U	30
Ctrl V ->	5	Ctrl D	576
↑	4687	Ctrl E	87
↓	5282	Ctrl Y	5
Ctrl V Ctrl E	1	i +	505
Ctrl C	464	Ctrl \ Q :	182
Ctrl z	25	\$	27
<-	6004	h	4281
->	7095	l	5208
Ctrl D	764	D	40
		dd	437
Ctrl R	1733	x	1248
Ctrl V Ctrl Z	1	o	56

* Averages are rounded to the nearest digit

+ This operation includes the "vi" commands
(A, a, I, R, C, and i)

"Ctrl V Ctrl E", have lower averages than the ones that need one command only.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS

This thesis consists of two major parts. The first part presents the editors that are available on the Unix at the CIS Department, Oklahoma State University. An introductory description of each editor is discussed to give a good start on the second part.

The second part presents the method that was used to collect the data for the editors' commands. Also an analysis of the data collected is discussed. The type of the data structure that was used to implement the monitor program is explained in appendix A.

It was found that over 95% of the monitored "ed" and "ued" work was performed by about 25% of the available commands, and over 93% of the monitored "ex" and "vi" work was performed by about 13% of the available commands.

It was also found that the average usage of most of "ex" and "vi" commands is much lower than "ed" and "ued" commands. This can be attributed to three reasons:

1. The number of commands available in "ex" and "vi" is almost 50% more than the number of commands available in "ed" and "ued".

2. It is possible to access "vi" from within "ex" and vice versa, while it is only possible to access "ed" from "ued" but not from "ed" to "ued". This resulted in forcing "ed" users to perform all the work using line mode commands.
3. The "vi" editor has a number of commands which are not available in "ued", such as "ZZ". This results in forcing "ued" users to change to line mode to perform the same operations.

The third and the final finding was that most of the operations which require two commands in order to be carried out, such as "Ctrl V" followed by "->", have much lower average usage than the operation which require one command only (see Appendix C).

Suggested Further Work

The result of this work leads to the following suggestions:

1. Since a small number of the total number of commands in both "ed" and "ued" and "ex" and "vi" editors performed over 93% of the total work, it is therefore, recommended that the source code for these commands should be examined in the hope that its efficiency can be improved, if possible.

2. The write and quit operation (one command) was heavily used in "ex" ("x" command) and in "vi" ("ZZ" command), while such command is not available in "ed" and "ued" (see Table XII, Appendix C) (the two commands "w" and "q" perform this operation), it is therefore suggested that adding such command to "ed" and "ued" would benefit the users. Similar argument can be presented for another "vi" command which is not available in "ued", this command is "." (repeat last modifying open/visual command).
3. The undo command "u" (discard last command) was used in "ex" and since "ex" and "vi" are linked together, therefore, this command is also available for the "vi" users. The "ed" editor does have the undo command ("u") while "ued" does not; therefore, it would be useful for "ued" users to add this command.
4. Unlike "ed" and "ued", the online manual entry for "ex" and "vi" does not have a detailed description of the commands. Also the learn command does not have an entry for "ex". It is therefore, recommended that adding these information to the system will give the CIS users a better environment to learn "ex" and "vi".
5. The "ued" command "Ctrl W" is used to widen a line to the right. This command is useful to insert character(s) between two characters within the same line. For example,

if a user needs to insert the word "elseif" in a line, he/she is likely to repeat "Ctrl W" six times or type "Ctrl V" 6 "Ctrl W" then types the word. Since the average usage of the "Ctrl W" was very high (1649 times per user during the monitoring period, see Table XI, Appendix C); therefore, the command "Ctrl V Ctrl E" is very useful in this situation. The command "Ctrl V Ctrl E" is not described in the manual, it is therefore recommended that should be added to the online manual.

6. Appendix C has a column containing the accumulative percentage values of the usage of the editors commands. This column was included to assist in any future simulation work which may be carried out in the CIS Department in the area of editors.

SELECTED BIBLIOGRAPHY

- (1) Bayer, R. and Unteraner, K. "Prefix B trees." ACM Transaction on Data Base System, Vol. 2, No. 1 (March 1977) 11-26.
- (2) Comer, D. "The Ubiquitous B Tree." Computing Surveys, Vol. 11, No. 2 (June 1979), 121-137.
- (3) Deutsch, P. and Lampson, B. W. "An Online Editor." Communication of ACM, Vol. 10, No. 12 (Dec 1967), 793-799.
- (4) Digital Equipment Corporation VAX-11, Text Editing Reference Manual, Maynard, Mass. (Aug. 1978).
- (5) Embley, D. W. and Nagy, G. "Behavioral Aspects of Editors." Computing Surveys, Vol. 13, No. 1 (March 1981), 382-385.
- (6) Fraser, C. W. "A Generalized Text Editor." Communication of ACM, Vol. 23, No. 3 (March 1980), 154-158.
- (7) Hansen, W. J. "Creation of Hierarchic Text with a Computer Display." Argonne National Laboratory, Argonne, III. Rep. ANL7818, (July 1971).
- (8) Irons, E. T. and Djourup, F. M. "A CRT Editing System." Communication of ACM, Vol. 15, No. 1 (Jan. 1972), 16-20.
- (9) Kernighan, B. W., A Tutorial Introduction to the Ed Text Editor, Bell Laboratories, (June 1982).
- (10) Kernighan, B. W., Advanced Editing on Unix, Bell Laboratories, (June 1982).
- (11) Kernighan, B. W. , Lesk, M. E. and Ossanna, J. F. "Unix Time Sharing System: Document Preparation." The Bell System Technical Journal, Vol. 57, No. 6 (July-August 1978), 2115-2135.
- (12) Kernighan, B. W and Ritchie, D. M., The C Programming Language, Prentice-Hall, 1978.
- (13) Knuth, D. E., The Art of Computer Programming Vol. 3:

Sorting and Searching, Addison Wesley, Reading, Massachusetts, 1973.

- (14) McMahon, Lee E., A Non interactive Text Editor, Bell Laboratories, (August 1978).
- (15) McGilton, H. and Morgan, R., Introducing the Unix System, McGraw-Hill, N.Y. 1983.
- (16) Meyrowitz, N. and Van Dam, A. "Interactive Editing System: Parts I and II." Computing Surveys, Vol. 14, No. 3, (September 1982), 321-415.
- (17) Parikh, J. S., "The Design of a Text Editor for VSAM Files." (Unpub. Master thesis, Oklahoma State University, 1980)
- (18) Ritchie, D. M. and Thomson, K. "The Unix Time Sharing System." Communication of ACM, Vol. 17, No. 7 (July 1974), 365-375.
- (19) Stearns, F. "How to Select a Text Editor." Interface, Vol. 7, No. 11 (Nov 1982), 108-118,164.
- (20) Unix Programmer's Manual. Vol. 1, Edition VII, 1984.
- (21) Van Doren, J. R. "Information Organization and Retrieval." (Unpub. Class Notes, Oklahoma State University. 1983.)
- (22) Walker, A. N. Private Correspondence. Nottingham University, England, Dec. 1983.
- (23) William, N. J., Ex Reference Manual, Bell Laboratories, (September 1980).
- (24) William, N. J., An Introduction to Display Editing with Vi, Bell Laboratories, (September 1980).

APPENDIX A

Prefix B+_Tree

The structure that is used to store counters for editor commands per user is a Prefix B+_Tree, described by Bayer et al. (1) and Comer (2).

Prefix B+_Tree is a special case of B+_Tree. In B+_Tree each node contains keys and pointers. The B+_Tree structure is divided into two levels (Figure 5). The upper level is called index. It contains keys and each key is copied from the bottom level key during a node split on insertion. The bottom level is called keys or leaf. It contains pointers which point to data record or external nodes. On the key level, there is only one pointer per key. Each leaf node has a link to the next right leaf node, except the most right node, which has a null link.

Some implementations of the B+_Tree (13) have data stored with the keys in leaf nodes; but in the one that is used in the monitor program, pointers to data are stored with the keys in leaf nodes.

In Prefix B+_Tree a key in the index level is the shortest separator to the leaf node when a node is split.

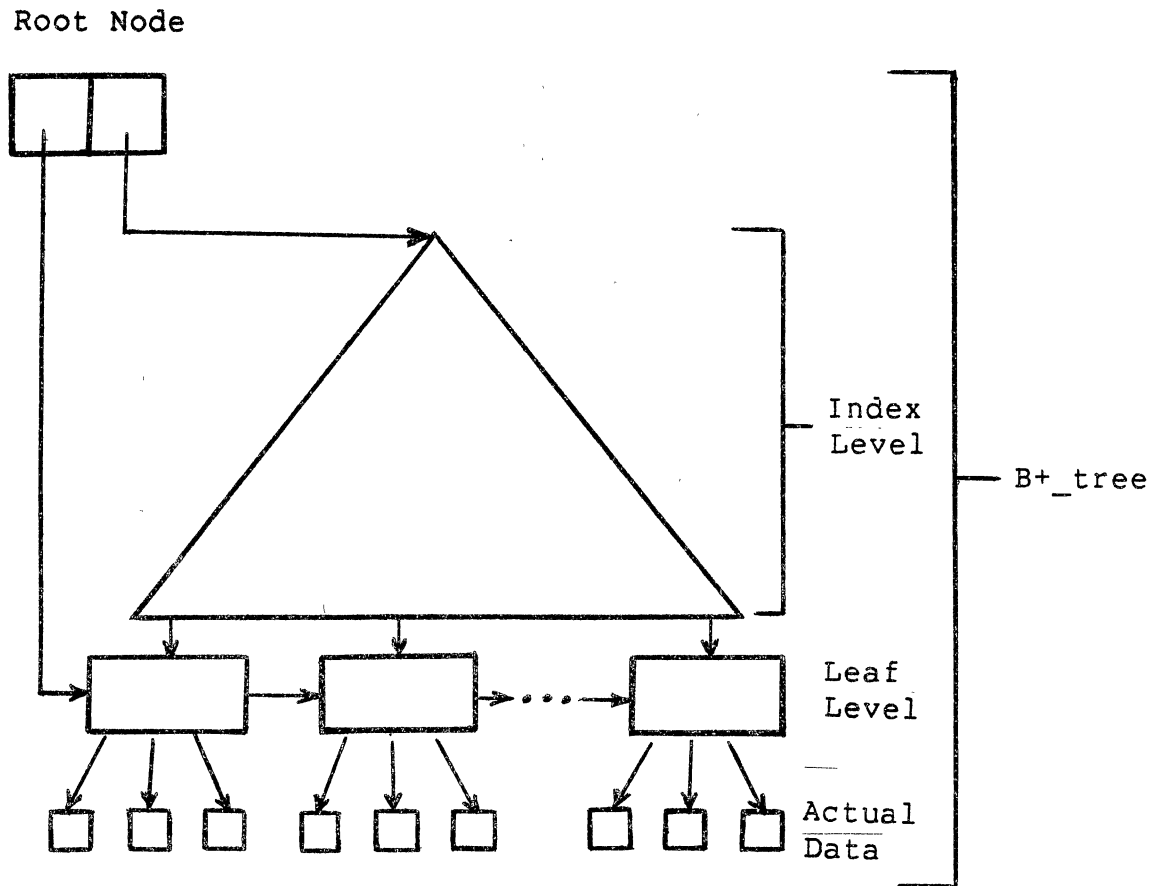


Figure 5. Organization of B+_Tree

The monitor program used a Prefix B+_Tree structure. This structure is stored in a file with a fixed record size. The programs were written in C and implemented on the Unix Operating System (12).

The Prefix B+_Tree file has three types of records (all have the same size) (21). The structure of the first physical record in the file is different from the remaining records, it is called the root (root node). Figure 6 shows the structure of this node, it contains two pointers: one points to the first leaf node and the other one points to the first index node. Figure 7 shows the structure of the index node. The structure of the leaf node is shown in figure 8.

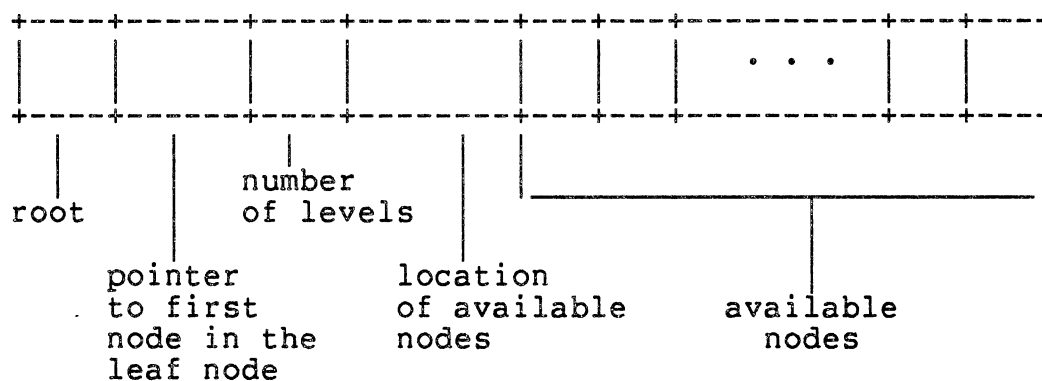


Figure 6. B+_Tree: Structure of Root Node

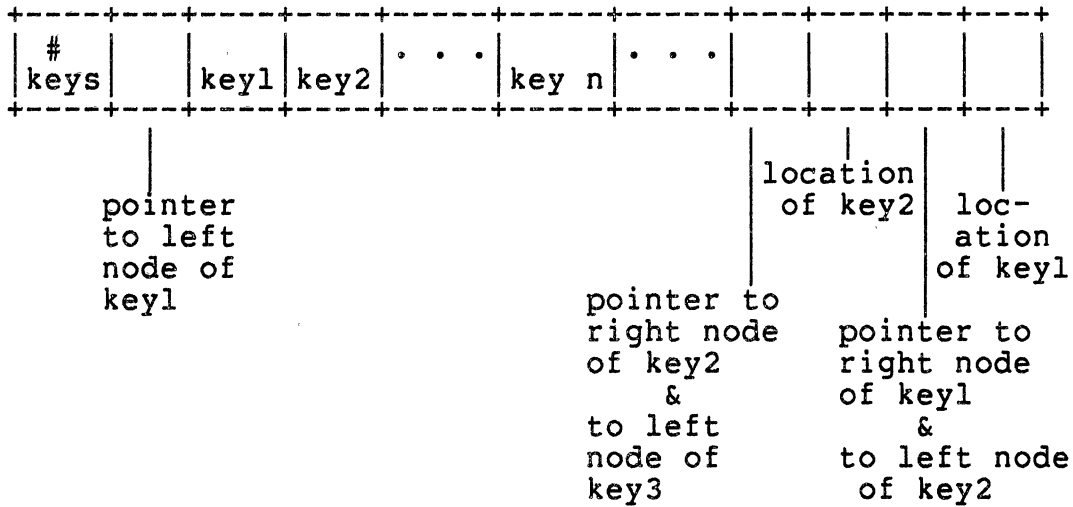


Figure 7. B+_Tree: Structure of Index Level Node

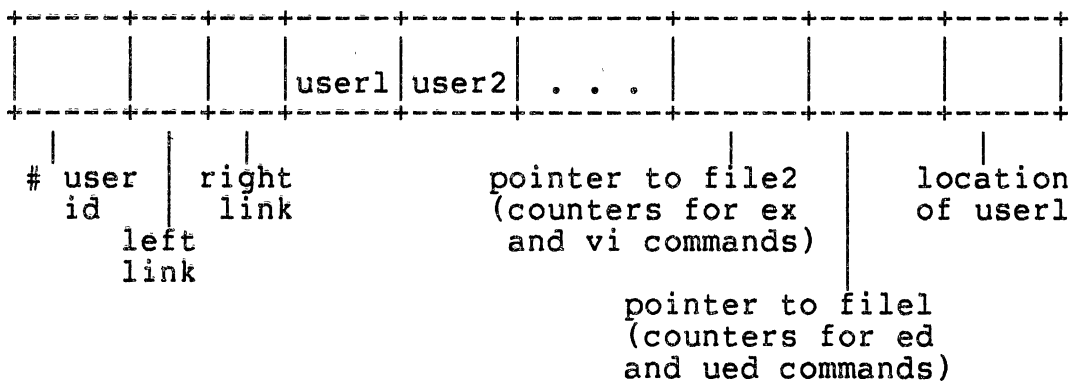


Figure 8. B+_Tree: Structure of Leaf level Node

APPENDIX B

EDITORS COMMANDS

TABLE VII

ED LINE EDITOR COMMANDS

Ed commands	Description
a	insert after (append)
b	backing over a read
c	change
d	delete line(s)
e	edit another file
f	print file name
g	global
i	insert before
j	join lines (default 2 lines)
k	mark line
l	print line(s)
m	move line(s)
p	print line(s)
q	quit
r	read file
s	substitute
t	copy line(s)

TABLE VII (Continued)

u	move to ued commands
v	global (for pattern not exist)
w	write (save file)
x	encrypt file
z T	print terminal type
=	print line number
" %	print visual line(s)
! #	quit temporary to shell commands
<	remove line number during editing
>	print line number during editing

TABLE VIII
EX LINE EDITOR COMMANDS

Ex commands	Description
a	insert after (append)
c	change
Ctrl d	print one page from current line
d	delete line(s)
e n	edit another file
f	print file name
g	global
i	insert before
j	join lines (default 2 lines)
k	mark line
l	print line(s)
m	move line(s)
o	open visual
p	print line(s)
q	quit
r	read another file
s	substitute
se	set terminal type
sh	quit to shell for more than 1 command
t	copy line(s)

TABLE VIII (Continued)

u	undo (discard last command)
ve	print version of the editor
vi	switch editor to visual mode (vi)
w	write (save file)
x	write and quit
y	yank
z	print visual
=	print line number
!	quit temporary to shell commands
<	shift line one tab to left
>	shift line one tab to right
#	numbering text
	print next line but one
* @	print content of register specified
"	named buffers

TABLE IX
UED FULL SCREEN EDITOR COMMANDS

command	Description
Ctrl A	add next line at the cursor position
Ctrl B	move to the beginning of the text
Ctrl C	exit from ued
Ctrl D	delete all characters right to the cursor
Ctrl E	move to the end of the text
Ctrl F	move cursor one left tab
Ctrl G	split line
Ctrl K (UP ↓)	move cursor one line down
Ctrl L (->)	move cursor one character right
Ctrl N	toggle space processing
Ctrl O	delete non space right to the cursor
Ctrl P	copy line
Ctrl Q	redisplay screen
Ctrl R	remove one character
Ctrl T	text "word" mode operation
Ctrl U	redisplay screen
Ctrl V {number}	repeats the "Ctrl_anything"
Ctrl V Ctrl E	insert and widen
Ctrl V Ctrl Z	move cursor to the beginning of the line
Ctrl V - Ctrl E	quit from Ctrl V Ctrl E

TABLE IX (Continued)

Ctrl V ↑	scroll down one page
Ctrl V ↓	scroll up one page
Ctrl V <-	scroll up half page
Ctrl V ->	scroll down half page
Ctrl W	widen
Ctrl Y	help: print at the bottom of the screen: file name, line number, and column number
Ctrl Z	move cursor to the end of the line
"RETURN"	move cursor to the beginning of next line
TAB (Ctrl I)	move cursor one tab right
<- (LEFT)	move cursor one character left
↑ (DOWN)	move cursor one line up

TABLE X
VI FULL SCREEN EDITOR COMMANDS

Command	Description
a	append at current cursor
A	append at end of line
b,B	back word
cc	change
C	change text (to end of line)
Ctrl B	redisplay screen with scroll down 1 page
Ctrl D	scroll down half page
Ctrl E	scroll down one line
Ctrl F	redisplay screen with scroll up 1 page
Ctrl G	display at the bottom of the screen file_name, current line #, and # of lines
Ctrl L	redisplay screen
Ctrl U	scroll up one page
Ctrl Y	scroll up one line
Ctrl z	suspend editor session's and temporarily return to shell
Ctrl ?	(delete or rubout) interrupts
Ctrl \	quit to command mode
Ctrl ^	return to previous file
Ctrl]	take word after cursor as a tag and then does the commands
dd	delete line

TABLE X (Continued)

D	delete to end of line
e	end of word
E	to end of following blank/nonblank word
h	back a character
H	move cursor to first line of screen
i	insert and shift the rest of the line
l, "	forward a character
I	insert at the beginning of line and widen line
J	join lines (default 2 lines)
L	move cursor to last line of screen
m	mark
M	move cursor to middle of screen
n	search to next match of current pattern
N	search to previous match of current pattern
o	insert after current line
O	insert before current line
p,P	print line(s)
Q	quit from visual mode
S	remove line and insert on it (overwrite)
u	undo last changing command
U	restore current line to initial state
w,W	word forward
x,X	delete a character
yy	yank lines to buffer

TABLE X (Continued)

Y	yank lines
z	from current line display #of lines specified
ZZ	write and quit
@	call Macros
.	repeats last modifying open/visual commands
~	change case of letter (upper or lower case)
&	same as & in command mode
:	read and execute command mode
)	next sentence
}	next paragraph
(back sentence
{	back paragraph
%	match () or {}
0	beginning of line
\$	to end of line
\	return to line specified by following mark
<<	shift line one tab to left
>>	shift line one tab to right
!!	filter through command
==	indent for LISP

APPENDIX C

USAGE OF EDITORS COMMANDS

TABLE XI

USAGE OF ED AND UED EDITORS COMMANDS

Command	Total	Accumulative
a	992	0.0005396
i	594	0.0008628
r	18612	0.0109873
c	447	0.0112305
d	4402	0.0136251
e	8	0.0136295
f	27	0.0136441
g	478	0.0139042
j	62	0.0139379
k	0	0.0139379
l	67	0.0139743
p	3781	0.0160311
m	600	0.0163575
t	356	0.0165512
q	19853	0.0273508
s	3948	0.0294985
u	23587	0.0423294

TABLE XI (Continued)

w	17922	0.0520786
" %	28	0.0520938
=	124	0.0521613
! #	901	0.0526514
b	15	0.0526596
v	32	0.0526770
x	57	0.0527080
T z	55	0.0527379
<	12	0.0527444
>	7	0.0527482
Ctrl A	598	0.0530735
Ctrl B	3366	0.0549046
Ctrl C	23645	0.0677670
Ctrl D	38962	0.0889616
Ctrl E	691	0.0893375
Ctrl F	3426	0.0912011
Ctrl S	3577	0.0931469
<-	306199	0.2597135
-> (Ctrl L)	361867	0.4565624
TAB (Ctrl I)	40501	0.4785942
↑ (DOWN)	269391	0.6251379
↓ UP (Ctrl K)	239034	0.7551679
"RETURN"	193828	0.8606067
Ctrl N	476	0.8608657

TABLE XI (Continued)

Ctrl Q	36	0.8608853
Ctrl P	5939	0.8641160
Ctrl O	2971	0.8657321
Ctrl R	88371	0.9138043
Ctrl G	29107	0.9296380
Ctrl T	1201	0.9302913
Ctrl U	1158	0.9309212
Ctrl V	23525	0.9437184
Ctrl V Ctrl E	6	0.9437217
Ctrl W	84084	0.9894618
Ctrl V Ctrl Z	12	0.9894683
Ctrl V - Ctrl E	6	0.9894716
Ctrl Y	78	0.9895141
Ctrl Z	1269	0.9902044
Ctrl V ↑	7501	0.9942848
Ctrl V ↓	10094	0.9997758
Ctrl V <-	148	0.9998563
Ctrl V ->	264	1.0000000

TABLE XII
 USAGE OF EX AND VI EDITORS COMMANDS

Command	Total	Accumulative
a	32	0.0000495
i	36	0.0001053
r	214	0.0004366
c	40	0.0004986
d	408	0.0011303
e n	183	0.0014136
f	116	0.0015932
g	42	0.0016583
j	67	0.0017620
k	0	0.0017620
l	14	0.0017837
p	710	0.0028830
m	20	0.0029140
t	23	0.0029496
q	1363	0.0050600
s	490	0.0058186
vi	12365	0.0249638
w	3633	0.0305889
z	10	0.0306043

TABLE XII (Continued)

=	203	0.0309187
!	160	0.0311664
ve	1	0.0311679
o	2	0.0311710
x	7842	0.0433131
<	0	0.0433131
>	1	0.0433146
#	3	0.0433193
sh	2	0.0433223
u	12	0.0433409
y	6	0.0433502
Ctrl d	11	0.0433673
!	0	0.0433673
* @	4	0.0433734
se	0	0.0433734
"	28	0.0434168
Ctrl L	1036	0.0450209
@	5	0.0450286
.	2355	0.0486749
Ctrl U	1298	0.0506847
Ctrl D	25327	0.0898993
Ctrl E	3839	0.0958434
Ctrl Y	229	0.0961980
m	62	0.0962940

TABLE XII (Continued)

Ctrl F	2820	0.1006603
Ctrl B	1897	0.1035975
z	70	0.1037058
Y	47	0.1037786
J	688	0.1048439
S	17	0.1048702
O o	9026	0.1188455
A a i R I	22230	0.1532649
~	22	0.1532989
Ctrl ?	323	0.1537990
Ctrl \ Q :	7984	0.1661609
ZZ	7880	0.1783617
P p	3862	0.1843414
Ctrl ^	4	0.1843476
Ctrl]	2	0.1843507
&	10	0.1843662
Ctrl G	96	0.1845148
Ctrl Z	0	0.1845148
u	1847	0.1873746
U	60	0.1874675
b B	1469	0.1897420
w W	3136	0.1945976
e	124	0.1947896
)	85	0.1949212
}	84	0.1950513

TABLE XII (Continued)

(38	0.1951101
{	35	0.1951643
E	150	0.1953965
%	70	0.1955049
0	2454	0.1993045
\$	1178	0.2011285
h	188341	0.4927430
l ^w	229136	0.8475218
D	1717	0.8501803
x X	54921	0.9352162
H	263	0.9356235
L	255	0.9360183
M	305	0.9364905
n	551	0.9373437
N	404	0.9379692
\	126	0.9381643
dd	19217	0.9679186
cc	18465	0.9965085
<<	238	0.9968770
>>	1579	0.9993218
!!	264	0.9997306
==	0	0.9997306
yy	174	1.0000000

VITA

Shakir Mahmood Hussain
Candidate for the Degree of
Master of Science

Thesis: A COMPARATIVE STUDY OF THE USAGE OF THE ED
AND EX FAMILIES OF EDITORS ON THE UNIX
OPERATING SYSTEM

Major Field: Computing and Information Science

Biographical:

Personal Data: Born in Baghdad, Iraq, May 4, 1953.

Education: Graduated from Al-markisiyah Secondary
School, Baghdad, Iraq, in June, 1972; received
Bachelor of Arts degree in Statistics from Al-
Mustansiriyah University, Baghdad, Iraq, in June
28, 1976; completed requirements for the Master of
Science degree at Oklahoma State University in
July, 1984.