USING THE COMPUTER TO COMMUNICATE: AN INTRODUCTION TO TEXT PROCESSING AT IIASA--THE *EDX* AND *NROFF* PROGRAMS

Michael M.L. Pearson

July 1980
WP-80-111

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
A-2361 Laxenburg, Austria

PREFACE

This paper is one of a series prepared by IIASA's Survey Project in cooperation with the Computer Services Department on the topic of using the computer to communicate. Our aim is to make the IIASA computer more accessible to IIASA scientists for a wide range of people-oriented activities: computer-based text editing and text formatting, decision support systems, computerized conferencing, multi-author manuscript preparation, and other operations promoting international team research among widely dispersed research teams. This particular paper talks mostly about the standard IIASA screen-oriented text editor program *edx* because *edx* is the starting point for anyone wishing to use the IIASA computer to communicate.

CONTENTS

INTRODUCTION, 1

PURPOSE, 2

LOGGING IN AND LOGGING OUT OF THE COMPUTER
GENERAL INFORMATION ABOUT USING THE COMPUTER, 2

Logging In, 2

Getting a Login Name and Password, 3

Finding a Terminal, 3

Using the Terminal, 3

Understanding Directories and Files:
Where You Are and What You Have Written, 5

Miscellaneous, 7

Setting Your Password, 7

Typing Ahead, 7

Erasing Errors on a Line, 7

CREATING A DOCUMENT
USING EDX, 8

The Utter Basics for Using EDX, 8

Making Directories and Moving
From One Directory to Another, 12

Beyond the Utter Basics of *EDX*, 14

Moving the Cursor Around
and Making Corrections, 14

Moving the Text Around, 21

Editing an Existing File, 24

Advanced Basics, 24

Left Character--the First
Command Described in the Manual, 25

More Commands from the *UNIX Programmer's Manual*, 27

WARNINGS, 35

NROFF -- HOW IT WORKS, 36

References, 40

USING THE COMPUTER TO COMMUNICATE: AN INTRODUCTION TO TEXT PROCESSING AT IIASA--THE *EDX* AND *NROFF* PROGRAMS

Michael M.L. Pearson

INTRODUCTION

In part, this working paper was commissioned by Computer Services to fill a pressing need at IIASA for introductory documentation on how to use the basic IIASA screen-oriented editing program, *edx*. *Edx* is the most common starting point for anyone interested in using the computer at IIASA, whatever his or her aim. It is a standard mechanism for putting data, text or programming code onto IIASA's computer. Hopefully, this document contains sufficient information so that anyone--even an entirely inexperienced user--can use *edx* for these purposes simply by reading and practicing what is written here.

Why put a tutorial in a working paper, the standard IIASA publications vehicle for research results? The answer has to do with our agreement to produce this guide in the first place. As explained earlier in this series on using the computer to communicate, the Survey Project is facilitating and evaluating team research among geographically dispersed scientists (see Pearson, 1980a). To accomplish this, documentation is needed on how to use hardware and software available at IIASA. Thus, in addition to supplying general information regarding the Survey Project's teleconferencing dissemination/evaluation activity, we are documenting from the ground up the tools that people cooperating with us will need to use (see Pearson, 1980b). *Edx*, embedded as it is in certain local IIASA computerized conferencing software, is one such tool.

This document will be supplemented by a forthcoming workbook (see Lathrop, forthcoming). We recommend it to anyone wishing a detailed review of the material covered here.

PURPOSE

This paper describes how to use IIASA's computer[1] to produce documents: papers, letters, memoranda and computer-mediated messages. It is primarily a tutorial and should be read while sitting at a computer terminal. Words appearing in square brackets, like this: [text processing], are terms discussed in a supplementary glossary currently being prepared for this series, and which will be available by September 1980.

This paper is designed to tell the beginner everything he or she needs to know in order to [log in] to and [log out] of the computer and create a preliminary document. It also briefly discusses how to [process] and print a [formatted], finished version of the document--a subject of one of the next papers in this series.

Contact IIASA Computer Services Department to learn if you have the current edition of this paper--this is the July 2, 1980 version.

This paper does not presume the reader to have any prior experience using computers.

LOGGING IN AND LOGGING OUT OF THE COMPUTER. GENERAL INFORMATION ABOUT USING THE COMPUTER.

LOGGING IN

To use the IIASA computer you need two things:

- a [login name] plus [password]

and

- a computer [terminal].

[1] IIASA's computer runs under the UNIX ™ operating system developed by Bell Laboratories. The procedures discussed in this paper are about UNIX and text-editing and text-processing programs running under UNIX. Since our paper is for people unfamiliar with computer systems, we shall make no further mention of UNIX, operating systems or other technical details. For those interested in such things we refer them to other papers in this series, to *A Tutorial Introduction to the UNIX Text Editor* by B.W. Kernighan and to *UNIX Time-Sharing System:* UNIX Programmer's Manual, Seventh Edition, Volume 1, January, 1979. All documents are available from the Computer Services Department.

Getting a Login Name and Password

A login name and password are the first things you must type on a terminal in order to get access to IIASA's computer system. Call IIASA's Computer Services Department to obtain them. In most cases your login name will be the same as your last name except that it is entirely in lowercase letters and limited to eight characters. Your password will be some [string] of alphabetic and/or numeric [characters] known only to you and used to prevent other people from accessing the computer under your name. As discussed later in this paper, you can change your password at any time by using the computer.

Finding a Terminal

Most computer terminals at IIASA are connected to the in-house computer system. There are two basic types of terminal: video (sometimes called [CRT]) and hardcopy (typewriter). Video terminals resemble television sets, whereas hardcopy terminals look more like electric typewriters. For going through the examples in this paper you should use a video terminal. Later you will want to use a hardcopy terminal for printing your formatted document. Except for [editing],* everything described in this paper can be done on terminals of either kind. Hardcopy terminals print on paper. Video terminals display on a TV-like screen and have a small square of light called a [cursor] that moves across the screen and indicates where you are on the screen while typing.

The largest number of generally available terminals are located in the terminal room, S-35 in the Schloss. Others are distributed around the institute. The use of some terminals is restricted the department, area or task to which they belong. Others are available to anyone wishing to use them. After you have found a free terminal you can tell that it is ready-to-use if the last line (bottom-most line that the terminal has printed) reads:

login:

Using the Terminal

In the examples of this paper bold face type indicates lines the computer prints; normal type face indicates what the user types. In this paper the symbol "↵" stands for a carriage return (marked on some terminals "return" on others "CR" or "newline".) See the glossary entry [notation] (Lathrop and Pearson, *Using the Computer to Communicate: A Glossary for Users of IIASA's Computer* forthcoming IIASA working paper) for more detail regarding terminal keyboards, [keys], and how we represent them in our examples.

In the example below a user with the login name "smith" logs on to the computer and then logs off by typing the word "bye".

---

*The word "editing" as used here is somewhat specialized. It simply means the adding or altering of text on a computer.

```
login: smith ⏎
Password:treehouse ⏎
% bye ⏎
login:
```

(In reality, the word "treehouse" does not appear on the screen when Smith types it. We make "treehouse" visable in our example here to indicate what Smith must type in order to access the computer.)

The % printed by the computer is called a [prompt.] A prompt is a character that the computer prints on your terminal indicating the computer is ready to work for you. This percent sign is the standard prompt of the IIASA computer. It means the user has successfully logged on and that the computer is prepared to accept [commands] from the terminal. Commands are words that tell the computer to do something; for example, "bye" is the command that tells the computer you wish to finish using it.

Most of the examples in this paper consist of commands and how the computer reacts to them. If you misspell a command the computer reacts by telling you that it cannot find the command. After that, it prompts again with % . When this happens you should just try typing the command again. For example, in the following example Smith logs on again and attempts to type "who"--a command which causes the computer to print out a list of people currently using the computer. Notice that Smith misspells "who" the first time and has to repeat the command. Also in this example a [broadcast message]--a short notice from Computer Services of interest to all IIASA users--is printed out before the first "% " appears. At the end Smith types--for no particular reason--a couple of carriage returns before typing "bye". Typing a carriage return in response to the prompt just causes another prompt to appear.

```
login:smith ⏎
Password:treehouse ⏎
The computer will be down today from 15:00 to 15:30.
% whho ⏎
whho: not found
% who ⏎
jim       tty8    Feb 18 10:40
budget    ttya    Feb 18 13:17
library   ttyi    Feb 18 09:32
fedra     ttyj    Feb 18 07:53
naki      ttym    Feb 18 09:02
smith     ttyt    Feb 18 13:07
castro    ttyv    Feb 18 13:09
% ⏎
% ⏎
% ⏎
% bye ⏎
login:
```

If after typing a carriage return you find that it takes more than several seconds for the new % to appear, this means that the system is [slow], i.e., heavily loaded. In such cases it may be advisable to log off and come back later when [response time] is better (shorter) and the system is more pleasant to work with. If you type a carriage return and nothing happens, 1) either the computer is terribly slow or not operating ([crashed]), or, 2) your terminal is in a peculiar state. In such cases you can call Computer Services, ext. 465 to find out whether it is the computer or your terminal that is to blame.

UNDERSTANDING DIRECTORIES AND FILES: WHERE YOU ARE AND WHAT YOU HAVE WRITTEN.

In the next section we shall decribe step by step how to create a document on the computer. Before doing this, however, it is important that the reader have an understanding of two basic concepts:

- Directory

and,

- File

Directories:

A [directory] can be thought of as a place with a name. As soon as you log in you are put automatically by the computer in a directory--usually one with the same name as your login name. While you use the computer you are always in a directory. Directories are hierarchical. They extend from one another like the branches of a tree.

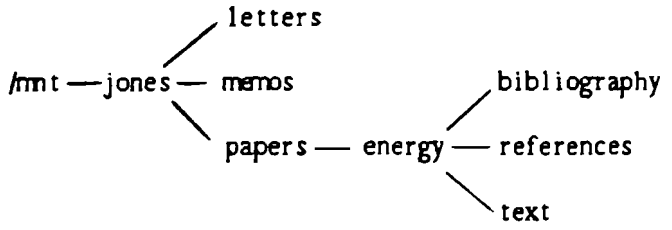
It is customary for users to make their own directories, move from one directory to another and use them to organize their work on the computer. We will talk about this in more detail later on. To find out at anytime where you are (in what directory you are currently located,) type the command [pwd]. The command "pwd" is an abbreviation for "print working directory." In the example below, Smith logs in and types "pwd" to find out where he or she is.

```
login: smith ↵
Password: treehouse ↵
% pwd ↵
/mnt/smith
```

The line **/mnt/smith** means that Smith is in a directory named "smith" which, itself, is a directory extending from the directory "mnt". The directory "mnt" is the usual place where the directories of new users are created ("mnt" is an abbreviation for the word "mount" a word used for historical reasons that need not concern us.) The directory "smith" is called Smith's [home directory]. It was created by Computer Services when Smith got his or her login name. If you do a "pwd" after first logging in it is possible (but unlikely) that you will find that your home directory has been created someplace other than in "mnt". Because you start automatically in your [home directory] each time you log in and build all your new directories from there, it is unimportant to you where your home directory has been created. Figure 1 shows a typical directory hierarchy. It extends from **/mnt/jones** and includes directories for letters, memos and a research pa-

per on energy which itself is subdivided into further directories.

Figure 1. A Directory Hierarchy



In reality directory hierachies with as many branches as the one above can be somewhat awkward to work with. There is no real limit, however, to the depth to which directory hierarchies can be extended in this fashion, (that is, how many directories can be created, one from the next.)

Directories are the basic mechanism for organizing your work on the computer. In the next section we will show how to make new directories, how to move from one directory to the next, how to remove directories and how to put [files] into directories.

Files:

A computer file is like pieces of paper. When you write a document, you enter the contents into a file which itself has a name of your own choosing. We recommend that file names be in lowercase letters. On the IIASA computer a file name can be no longer than fourteen [characters].

The command [f] gives the names of any files in the current directory along with the names of any directories that extend from the current directory. Since Smith has just logged in for the first time, has no files and has made no new directories, typing "f" produces nothing very exciting:

```
% f ↵
%
```

We will talk more about *f* below as Smith makes files and additional directories. What *f* has to say will be much more interesting then. Remember that the commands *pwd* and *f* are ways to find out at anytime where you are and what files you have in this "place," that is, in your current directory.

Smith will write a note to a colleague by entering his text in a file called "jonesnote" using the editing command [edx]. "*Edx*" is the common way to use video terminals to create files and put text into them. *Edx* cannot be used on hardcopy terminals.

MISCELLANEOUS

Setting Your Password

If you wish to change your password you can create a new one at anytime by using the command [passwd]. In the example below Smith will log in twice, once with the old password "treehouse" (before setting the new one) and then again with the new password "bobsled".

```
login:smith ↵
Password:treehouse ↵
% passwd smith ↵
New password:bobsled
Retype new password:bobsled
% bye ↵
login:smith ↵
Password:bobsled ↵
%
```

Notice that for the command *passwd* to work it must be followed on the same line by the name of the user, in the example above, by "smith". When a command requires that additional information be typed on the same line together with it, the command is said to take an [argument]. Many commands take arguments and we will encounter more of them later in this paper. Remember that in reality the word "bobsled" (Smith's new password) would not appear on the screen of a video terminal nor be printed by a hard-copy terminal. Normally when you type, everything is [echoed], that is to say, printed by the terminal after you strike the key. For passwords , since they are secret, this is not the case.

Typing Ahead

The prompt "% " means that the computer is ready to [accept input.] For now you should wait for the prompt to appear before typing anything new. Experienced users often [type ahead,] anticipating what the computer will do and not waiting for the prompt before entering new commands. For a clear understanding of the information contained in this paper we recommend that you do not type ahead but wait for the prompt to appear before doing anything.

Erasing Errors on a Line

If at anytime while typing a line on the computer you wish to erase a character you may do so by striking the [delete] key. On most IIASA terminals this key is marked "del" and must be depressed while also depressing the "shift" key. On some terminals the "delete" key is called the [rubout] key and may be marked "rub" or "rubout". Similarly you may type <control-u> to delete the entire line you are currently typing. The <control-u> is done by typing a lowercase *u* while simultaneously depressing the key on your terminal marked "control". You can delete characters or lines in these two ways at any time while using the computer--while typing your login name, password, commands or text for a document. While discussing the editing program ***edx*** in the next section we shall encounter these deleting mechanisms again. Delete and <control-u> are

unusual because they are functions that you can use at any time while typing on the computer. Most of the things described below pertain only to *edx* and will only work when you are using the *edx* editing program. Most will not work when the computer is prompting with the % sign. See Lathrop, *Using the Computer to Communicate: A Text Processing Workbook*, a forthcoming paper in this series for more discussion of these matters.

CREATING A DOCUMENT USING EDX

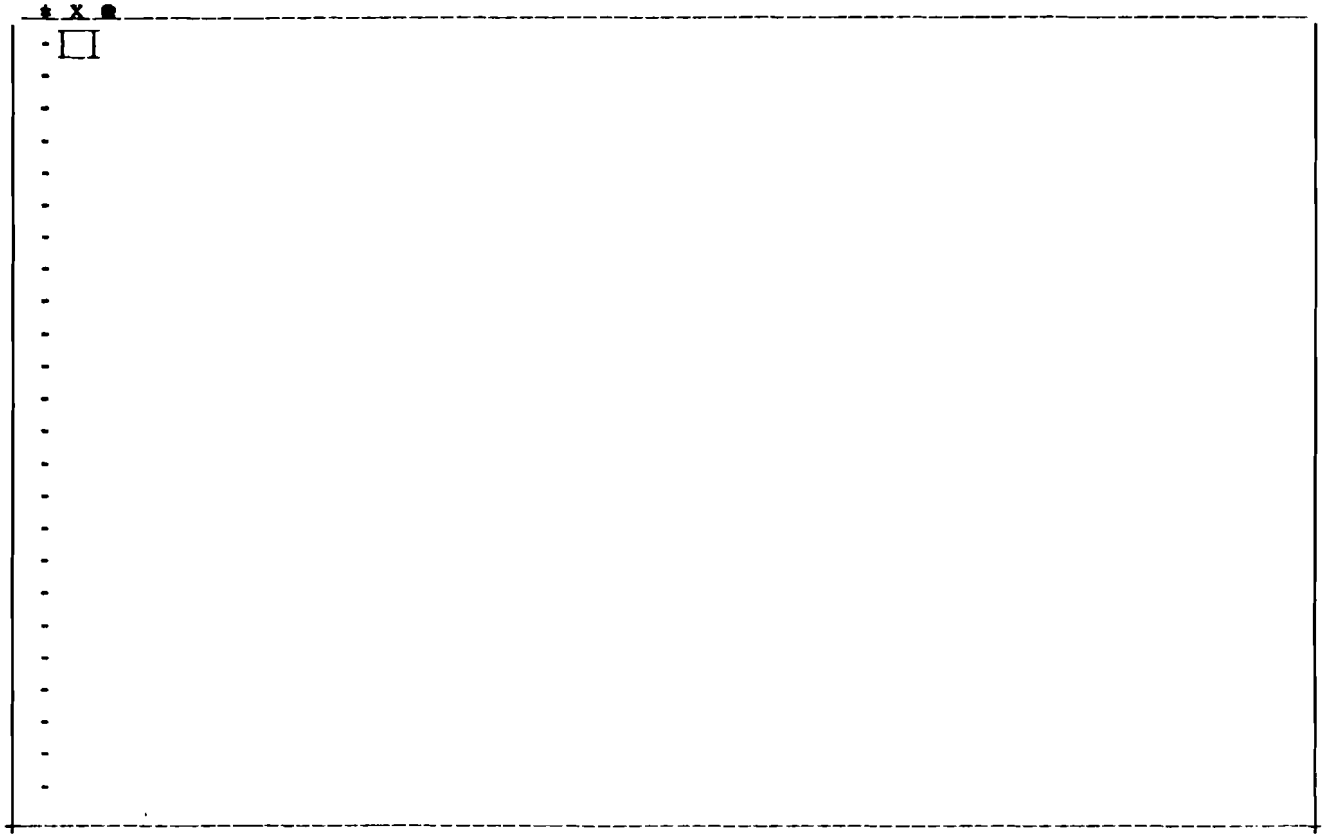
This section of the paper describes how to write text in a file using the command *edx*. *Edx* is a powerful editing [program] with many features for deleting, inserting, modifying and moving text. Although we will talk about major features of *edx* before the end of this section, we will start with the absolute minimum needed to know in order to put simple text on the computer.

THE UTTER BASICS FOR USING EDX

Smith has just logged in and wants to use the computer to compose a note to Jones. Smith should probably make a special directory for messages and go there to create the file containing this message. For now, however, let us assume Smith wants to create the message in the home directory **/mnt/smith** where he or she is located immediately after first logging in.

To use the editing program, Smith will type "edx" followed by an appropriate name for a file which is to contain the text. Here the name "jonesnote" is choosen. File names can be no longer than fourteen characters. It is recommended that they consist solely of lowercase letters and/or numbers.

```
login:smith ●
Password:bobsled ●
% edx  jonesnote ●
?jonesnote
* x ●
```



The computer prints "?jonesnote" because no file named jonesnote exists in Smith's current directory. (Later we will talk about what happens if the file does, in fact, already exist, but for now let's not get ahead of ourselves.) The "*" is the prompt you get when first using *edx* on a new file. Next Smith types "x" followed by a carriage return to start the "magic" of "*edx*"s full-screen display. After typing "x" and a carriage return anything that appeared on the screen before the "x" was typed is erased. The entire screen of the terminal now looks something like the display we have enclosed in a box in the example above. After typing "x" the screen's left-hand margin becomes a column of hypen characters and the [cursor] appears at the upper left-hand corner. In our *edx* examples we will represent the cursor with the symbol []. For our non-*edx* examples we don't bother to indicate the cursor. At this point (after typing "x" followed by a carriage return and getting *edx's* full screen display) just typing text and doing carriage returns is all it takes to write text on the computer.

Below we see the results of Smith having carefully typed two error-free lines and then mispelling the word "research" on the third line. Note that as Smith begins a new line, i.e. strikes any key after doing a carriage return, the hypen at the left-hand margin of the line where the cursor is located automatically becomes a vertical bar char-

acter "|". The hyphens indicate that the lines are empty and not part of the file.

```
|To: B.Jones ←
|From: D.Smith ←
|Subject: Proposed joint researcb□
-
-
-
-
-
-
-
```

Since Smith has not yet done a carriage return, but is still on the line with the mispelled word "researcb", it is possible for he or she to correct the mistake by merely striking the delete key. Below Smith uses the "delete" key to correct the misspelling. We use [DELETE] to indicate where Smith depressed the delete (or rubout) key. When the delete key is depressed, the cursor on the screen moves to the left and the unwanted character disappears. You can depress "delete" as may times as you need to. Depressing "delete" or "rubout" once the cursor has backed up to the left-hand margin will simply cause your terminal to beep harmlessly at you. Depressing the delete key longer than a second or two has on some terminals the same effect as repeatedly striking the key rapidly.

```
|To: B.Jones ←
|From: D.Smith ←
|Subject: Proposed joint researcb[DELETE]h project□
-
-
-
-
-
-
-
```

Having finished typing the text, Smith depresses first the [escape] key--marked "esc" or "altmod" on the terminal keyboard--and then "q" (for quit.) Typing these two keys* sequentially--not simultaneously--terminates the full-screen display of *edx* and shows this fact by prompting with "*". Next Smith types a lowercase "w" (for write) followed by a carriage return to make a permanent copy of the contents of the file "jonesnote" on the computer. The computer responses by printing out the number of

*Use of the "escape" key is different from that of the "control" key. The "escape" key must be depressed *before* the "q" key, whereas "control" is always depressed *simultaneously* with another key. The notation <escape...q> is used here to indicate that the "escape" and "q" keys are typed sequentially. Note that the words "escape q" will not appear on the screen as you type an <escape...q>.

characters in the file "jonesfile", in this case one hundred sixty three. Smith types "q" to quit the editor program.

```
|To: B.Jones
|From: D.Smith
|Subject: Proposed joint research project ●
| ●
|Having talked with Roberts I now believe it is OK to ●
|go ahead with the project as planned.<escape...q>
-
-
-
-
-
-
-
```

```
*w ●
163
q ●
%
```

As a result of the above, there now exists a file "jonesnote" in Smith's current directory. Typing [f] confirms this fact:

```
% f ●
jonesnote
%
```

Just as [pwd] confirms the fact that Smith is currently in the directory "smith":

```
% pwd ●
/mnt/smith
%
```

The file "jonesfile" is in the directory "smith" because that was where Smith was located at the time he or she typed the command *edx*.

The standard way to print out the contents of a file after you have created it is by using the program [sl]--smart list--a program for quickly listing the contents of files. Whether you have logged in using a video terminal or a hardcopy terminal, *sl* is the best way to see in a hurry what a file contains. Type "sl" followed by the name of the file you wish to see. Of course you must be in the directory where the file is located.

If you are using a video terminal when you type the *sl* command, and the file has more than one screen-full worth of material in it (24 lines), the computer will pause after each screen-full. To see the next screen-full depress the carriage return key once. In the example below Smith prints out the complete contents of "jonesnote".

```
% sl jonesnote ⏎
To: B.Jones
From: D.Smith
Subject: Proposed joint research project

Having talked with Roberts I now believe it is OK to
go ahead with the project as planned.
%
```

We have described the minimum that you need to know in order to put text in a file on the computer. By typing very carefully and using the "delete" key it is theoretically possible to write hundreds of lines on the computer using *edx*. The power of *edx*, however, lies in its additional features, in particular its cursor control and text modification functions. Using these it is possible not only to create new files but to edit (modify or change) existing ones. In *edx* it is simple to back up and go forward within a file and modify its contents. The power of *edx* will be the subject of the section after next.

MAKING DIRECTORIES AND MOVING FROM ONE DIRECTORY TO ANOTHER

Let us say that Smith wants a directory called "messages" where notes such as the one he or she wrote in the file "jonesnote" can be kept separated from other material stored on the computer. To do this Smith uses the "make directory" command [mkdir].* Since Smith is currently in **/mnt/smith** typing...

```
% mkdir messages ⏎
```

creates the directory "messages" under "smith". Now "messages" shows up when Smith uses the *f* command. Recall that *f* lists all the files in the current directory and also lists any directories that extend from the current directory.

```
% f ⏎
jonesnote
messages
%
```

The *f* command** does not tell you which of the two names "jonesnote" and "messages" is

---

*Conversely there exists a "remove directory" command [rmdir] that will remove a directory which is empty, that is, contains no files or subdirectories. Understanding how to use the *mkdir* command described here is all you need to know in order to use *rmdir*.

**One can use the "long form" of the *f* command by typing *f –l* instead of *f* in order to see if the names are those of files or directories. (Note that the "*–l*" is a "minus and lowercase letter l" and not "minus one".) After typing *f –l* a *l*ong line of information about each file and/or directory will appear on the screen--one line per name. If the very first character of a line is the letter "d" then the name is that of a directory. In all other cases the name is that of a file.

a directory and which is a file. It is recommended that you don't forget directory names. At present, if you make a mistake and try to edit a directory, you get very strange results.

Now it is possible for Smith to move to the directory "messages" by typing the "change directory" command [cd]...

```
% cd messages ●
```

And typing *pwd* shows that Smith has, indeed, moved to a new directory.

```
% pwd ●
/mnt/smith/messages
%
```

This use of *cd* is, in fact, what Smith should have done before making the file jonenote.

The command *cd* can take a full pathname such as **/mnt/smith** as an argument. Thus to go back to the [parent directory] of "messages" (the directory from which "messages" extends) Smith can type...

```
% cd /mnt/smith ●
```

For moving around quickly from directory to directory* it is useful to know that the *cd* command takes as an argument a shorthand representation--two dots—for "parent directory". Thus, had Smith typed...

```
% cd .. ●
```

while in **/mnt/smith/messages**, he or she would have achieved the same result as typing...

```
% cd /mnt/smith ●
```

The two dots may be used at any time to stand for "parent directory". For example, the command...

```
% cd ../.. ●
```

is perfectly acceptable. If done while in **/mnt/smith/messages** it will move Smith to **/mnt**.

Finally, the "remove file" command [rm] removes files. Typing...

```
% rm jonesnote ●
```

will erase the file "jonesnote" if Smith types it while he or she is in the directory in which

*There are some much more powerful tools available for moving from directory to directory which are not described here. For details see documentation on the UNIX C-shell command interpreter program available from Computer Services.

"jonenote" is located, that is, in the directory **/mnt/smith**.

For more information on moving, copying, removing, renaming files and directories, see the last section of this paper (see also Lathrop).

BEYOND THE UTTER BASICS OF *EDX*

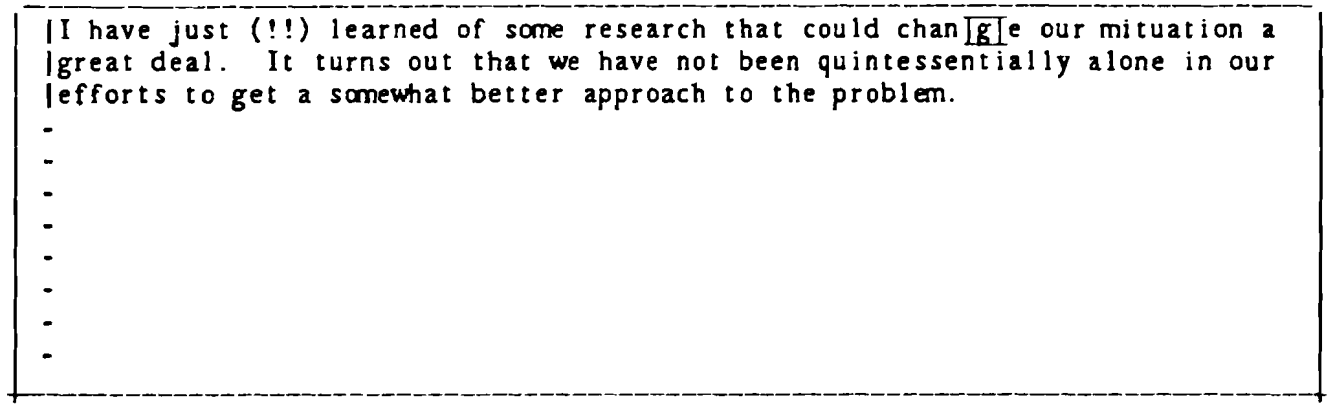
After typing the command "edx jonesnote" Smith was able to add text to the file "jonesfile" by using the terminal more or less as a standard typewriter. Depessing any key on the terminal that resembles a key on a standard typewriter--letter, number or punctuation mark--causes that letter, number or punctuation mark to appear on the screen and the cursor to move one position to the right--providing that the cursor has not reached the far right-hand side of the screen where it can go no farther and where the terminal merely beeps each time a key is depressed. Note that by holding down the key marked "control" and typing another (appropriate) key at the same time it is possible to do more than just duplicate a typewriter. For example, it is possible to move the cursor around on the screen without disturbing text. Below we see a situation where this is useful.

Moving the Cursor Around and Making Corrections

In the example below Smith begins a file "robertsnote" using *edx* and after typing three lines of text notices that the first line contains the misspelling "mituation."

```
% edx  robertsnote ⏎
?robertsnote
x ⏎
|I have just (!!) learned of some research that could change our mituation a
|great deal.  It turns out that we have not been quintessentially alone in our
|efforts to get a somewhat better approach to the problem[.]
~
~
~
~
~
~
~
~
```

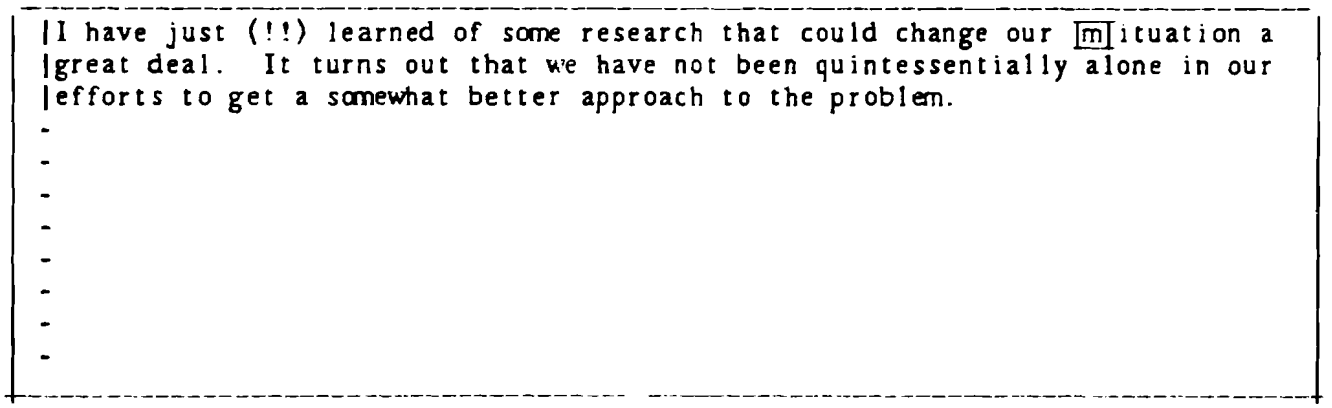
By holding down the "control" key and typing "k" twice (typing twice what we represent by <control-k> in this paper) Smith moves the cursor up two lines, that is, two vertical positions.



The letters "h", "j", "k", and "l" when typed with the control key depressed, that is, as <control-h>, <control-j>, <control-k> and <control-l>, move the cursor one position: left, down, up and right respectively.

<control-h>	move the cursor one position to the left
<control-j>	move the cursor one position down
<control-k>	move the cursor one position up
<control-l>	move the cursor one position to the right

Thus, by typing <control-k> twice to move the cursor up two lines and typing <control-l> seven times to move the cursor to the right, Smith puts the cursor on top of the letter "m" in the typo "mituation."



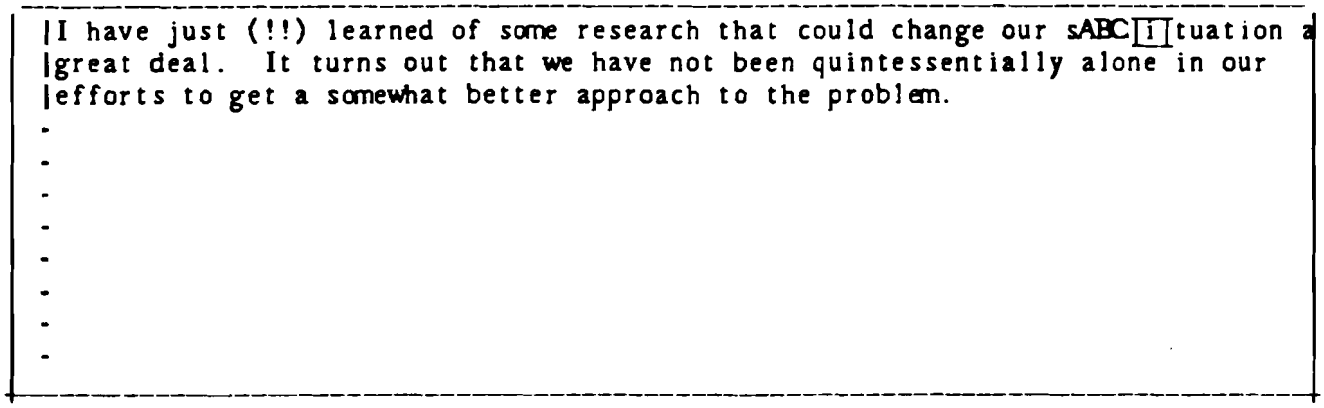
The <control-f> command removes the character the cursor is positioned on. When Smith types a <control-f> the unwanted letter "m" is deleted and the screen now looks like:

```
|I have just (!!) learned of some research that could change our [i]tuation a
|great deal.  It turns out that we have not been quintessentially alone in our
|efforts to get a somewhat better approach to the problem.
-
-
-
-
-
-
-
-
```

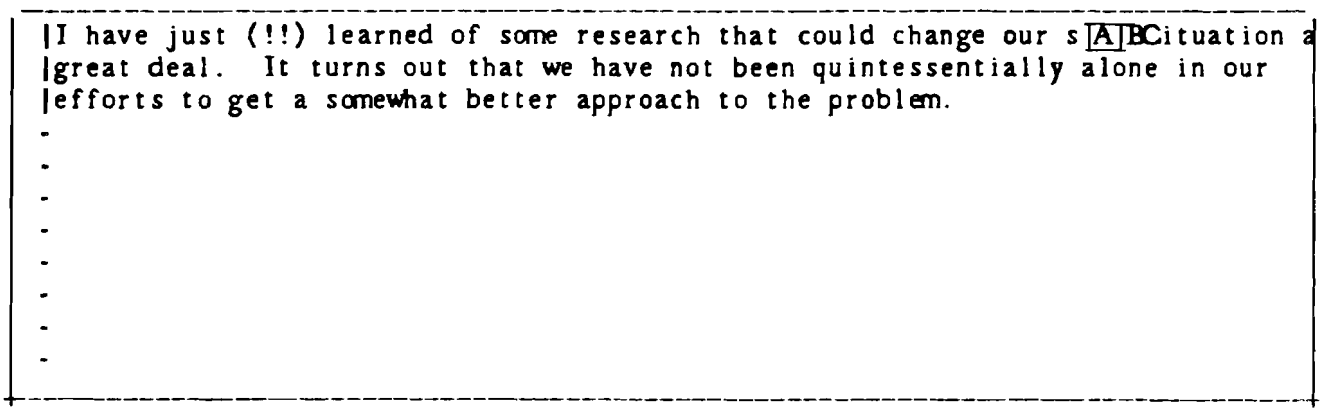
And if Smith types an "s" (without holding the control key!), the misspelling is corrected.

```
|I have just (!!) learned of some research that could change our s[i]tuation a
|great deal.  It turns out that we have not been quintessentially alone in our
|efforts to get a somewhat better approach to the problem.
-
-
-
-
-
-
-
-
```

Typing "s" inserted the letter "s" in front of the character the cursor was positioned on before the key was depressed. In the example above "s" is inserted in front of the letter "i". Notice that this is a so-called [insert mode] of operation. For example, if Smith now types the uppercase letters "ABC" they appear in front of the character the cursor is positioned on. The line to the right of the inserted characters is pushed further to the right.



The recommended sequence for removing unwanted characters is to put the cursor on the left-most character of the [string] of characters you want to remove--in the example below the string "ABC"...



and to remove the unwanted characters using <control-f> as many times as needed, and then, if necessary, to type in the desired corrections (in this case it is not necessary.)

```
|I have just (!!) learned of some research that could change our s[i]tuation a
|great deal.  It turns out that we have not been quintessentially alone in our
|efforts to get a somewhat better approach to the problem.
-
-
-
-
-
-
-
-
```

Moving the cursor one position at a time is a slow way to work in *edx*. In order to move the cursor more quickly, there are the two commands:

<control-o>	move the cursor to the beginning of the word (or string of punctuation) on the left
<control-p>	move the cursor to the beginning of the word (or string of punctuation) on the right

These are the fast versions of <control-h> and <control-l>. Notice were the cursor is located after Smith types seven <control-o>'s in a row:

```
|I have just (!!) learned of [s]ome research that could change our situation a
|great deal.  It turns out that we have not been quintessentially alone in our
|efforts to get a somewhat better approach to the problem.
-
-
-
-
-
-
-
-
```

Three more <control-o>'s put the cursor on the first parenthesis.

```
|I have just [{[!!) learned of some research that could change our situation a
|great deal.  It turns out that we have not been quintessentially alone in our
|efforts to get a somewhat better approach to the problem.
-
-
-
-
-
-
-
-
```

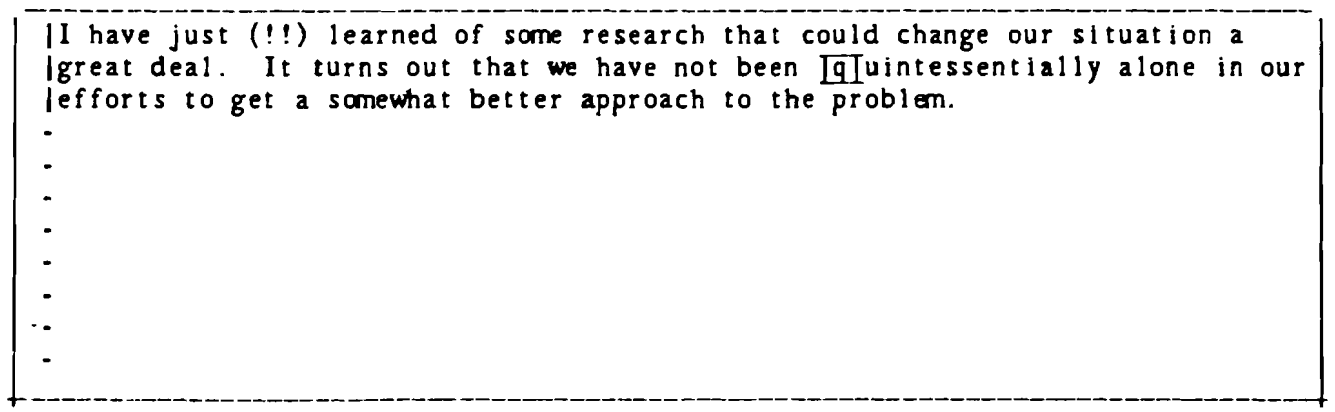
Another way to move the cursor quickly is with "carriage return." Depressing carriage return in ***edx*** is like typing <control-h> except that the cursor jumps to the beginning of the next line regardless of the cursor's previous horizontal position. Carriage return is a convenient way to move the cursor down the screen because unlike <control-h> you don't have to hold down the control key while depressing it. Also, it's often desirable to have the cursor at the left-hand margin where you can easily find it.

A <control-g> puts the cursor in the upper left-hand corner of the screen and <control-n> moves the cursor to the end of the line it is located on. You'll find that:

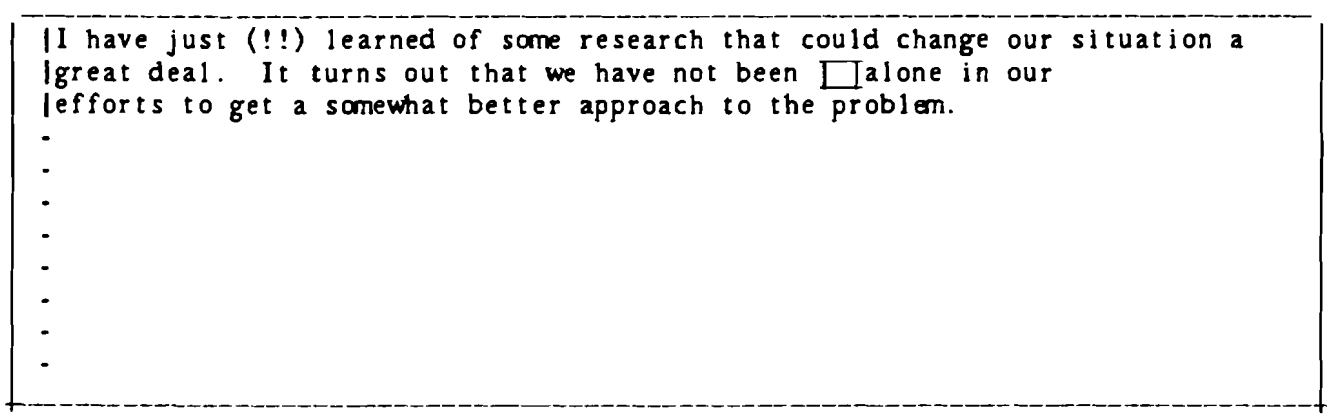
<control-o>	move the cursor to the beginning of the word (or string of punctuation) on the left
<control-p>	move the cursor to the beginning of the word (or string of punctuation) on the right
<control-g>	move cursor to the top left-hand corner of screen
<control-n>	move cursor to end of current line
carriage return	cursor one position down and to the left margin

are the standard commands for moving the cursor around the screen in a hurry.

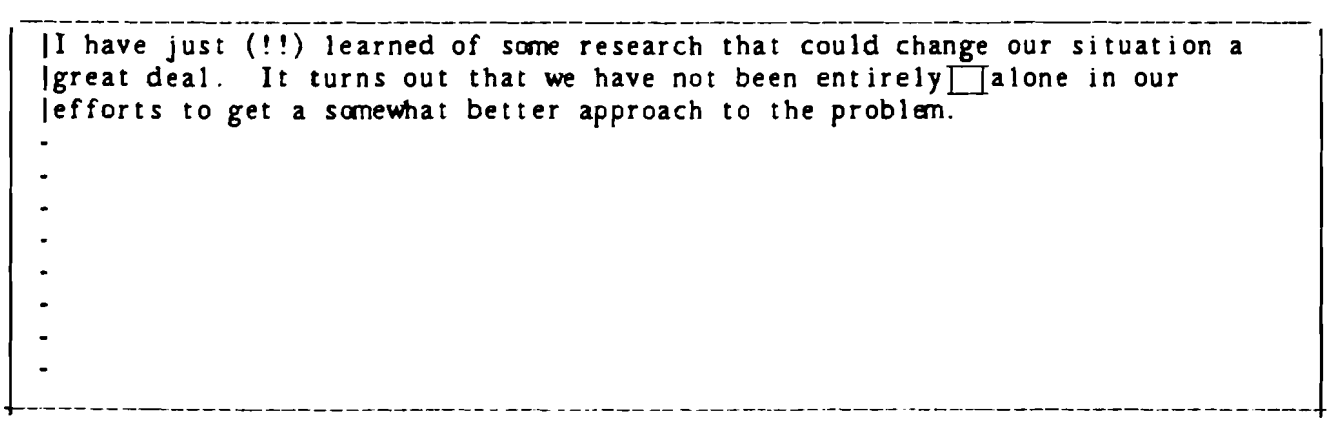
For removing typing errors in a hurry the <control-d> command is the fast version of <control-f>. Instead of typing sixteen <control-f>'s to remove the word "quintessentially" for example, Smith needs only to move the cursor to the first letter of the word...



type <control-d> once...



and then type the desired replacement text—in this case the word "entirely". Thus <control-d> removes all characters to the right of the cursor until the end of the word.



It was not necessary to type spaces (depress the space bar) before or after typing the word "entirely". They are still there, not removed by Smith's having typed the <control-d> to remove "quintessentially".

Moving the Text Around

Knowing how to modify anything on the screen in *edx* means it is easy to create files containing many lines of error-free text even if you are a bad typist. How does one type more than a screen-full of text? You have noticed perhaps that if you start typing in a file using *edx* and reach the bottom of the screen, the text [scrolls] upward one line a a time each time you do a carriage return.

In the example below Smith has almost reached the bottom of the screen:

```
|I have just (!!) learned of some research that could change our situation a
|great deal.  It turns out that we have not been entirely alone in our
|efforts to get a somewhat better approach to the problem.
|
|Note as well the following list of people whose work
|we have ignored to-date:
|
|                Appleby
|                Yorque
|                Lindsay
|                Edwards
|                Robertson
|                Lowery
|                MacKinnon
|                Cohen
|                Petrov
|                Arnold
|                Michaels
|                Henderson
|                Mathrop
|                Burns
|
|I my opinion if we fail to take account the efforts of these
```

By typing <control-w> Smith can move the line the cursor is positioned on to the top of the screen. This is exactly as if the text itself were pulled upwards as a single sheet of paper or, using another analogy, raised like a window shade, with the terminal screen the window. Below is the result of Smith's having typed <control-w>:

```
|I my opinion if we fail to take account the efforts of these[ ]
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
-
```

Conversely, a <control-e> drags the line the cursor is positioned on to the bottom of the screen. Below is the result of Smith's next having typed <control-e>:

```
|I have just (!!) learned of some research that could change our situation a
|great deal.  It turns out that we have not been entirely alone in our
|efforts to get a somewhat better approach to the problem.
|
|Note as well the following list of people whose work
|we have ignored to-date:
|
|                Appleby
|                Yorque
|                Lindsay
|                Edwards
|                Robertson
|                Lowery
|                MacKinnon
|                Cohen
|                Petrov
|                Arnold
|                Michaels
|                Henderson
|                Mathrop
|                Burns
|
|I my opinion if we fail to take account the efforts of these
```

The cursor could not be dragged all the way to the bottom in this example because there is not enough text. If this file were more than twenty four lines long and the cursor located any place below line twenty three then the line would have gone to the bottom.

Notice that another <control-e> at this point does nothing. The line the cursor is on is at the bottom and can go no further.

If you wish to flip forward or backward in a large file a screen-full at a time, use the commands:

<control-v>	move forward one screen-full (think of the German word "[V]orwaerts")
<control-b>	move backward one screen-full (think of the word "[B]ackwards")

Typing <control-v> displays your next twenty-four lines of text, <control-b> displays the twenty-four lines of text preceding where you are. Using the commands <control-v> and <control-b> is like turning the pages in a book. Unless you are are at the end or the beginning of the file, <control-v> and <control-b> give you an entirely new screen-full of text.

Editing an Existing File

To continue editing an existing file, say, "robertsnote" even after having logged out and then back in, Smith need only be in the directory where "jonesnote" is located and type...

```
% edx jonesnote ↵
```

Because the file already exists, this command will automatically initiate *edx's* full-screen display and put Smith on the first line of the file. The * (asterisk prompt) will not appear and Smith will not have to type "x" followed by a carriage return.

ADVANCED BASICS

A full explanation of all the features of *edx* is given in the *UNIX Programmer's Manual*. Since the description there is complete but difficult to read for people unfamiliar with computers, our approach in this section will be to take excerpts from the *UNIX Manual* and supplement them with our own commentary. We hope by this to add more useful commands to the reader's repertoire and, more importantly, offer insight into how to read the terse descriptions of the *UNIX Programmer's Manual*, thus opening the full power of *edx*.

Left Character--the First
Command Described in the Manual

After a short description of *edx* the section of the *UNIX Programmer's Manual* entitled "EDX(1)" presents a list of all the program's commands under the three headings: cursor movement, screen movement, and text modification. Note that in the *UNIX Manual* "control commands" are represented only by the brackets "<>". This notation is different from ours. For example, a "control b" which we indicate with <control-b> is represented by <b> in the *UNIX Programmer's Manual.*

Lets us begin by taking the *UNIX Manual's* first example under CURSOR MOVEMENT COMMANDS. This is the one that talks about "control h," the "move to the left" command--what we would symbolize with <control-h>.

Left Character

<h>	Move left one character
<c> <h>	Repeat search left using previous *string* argument
<c> *number* <h>	Move left *number* characters
<c> *string* <h>	Search to the left for the next occurence of *string*

The heading "Left Character" is short for "move the cursor one character-position to the left." The line...

<h> Move left one character

means "depress <control-h> and the cursor moves one character-position to the left." This, of course, is true until the cursor reaches the left-hand margin and can go no further.

"Left Character" can do more that just move the cursor to the left one character at a time. It has other powers when used together with <control-c>. For example, it will permit you to search to the left of where the cursor is located for some sequence of characters on the current line. This could be useful if you are doing accounting and your text consists of many rows of numbers. Otherwise it is not a very useful command since you can usually find a string of characters faster by looking for it.

The line in the *UNIX Manual* that reads...

<c> <h> Repeat search left using previous *string* argument

only makes sense if first you understand the line...

<c> *string* <h> Search to the left for the next occurence of *string*

which means:

> Depress <control-c>. The letters "Arg:" along with the cursor will suddenly appear in the upper left-hand corner of your screen. This is something which happens every time you depress <control-c> in *edx*; it means the program is ready to accept an [argument], that is, some qualifying information for another command

you intend to use--in this case <control-h>; don't worry about "Arg:" appearing up there; it has no effect on the text you've written and disappears as soon as you depress the carriage return key or the other command you are using it with. When the "Arg:" disappears, note as well that the cursor will jump back to where it was before you depressed <control-c>. Type in a string of characters that you want *edx* to search for then depress <control-h> and the cursor will move to the left to the beginning of that string, providing of course, that the string exists on the line. If the string you asked <control-h> to look for is not there, *edx* will beep at you.

Note also that if the string you are looking for starts with a numeric character instead of an alphabetic character (you're looking for "89274" not "Smith") then you must type a horizontal space (depress the space bar) before typing in the string. Typing in a number after <control-c> without a space prepended is a special case and we will talk about it in a minute.

Now knowing these facts, the line...

<c> <h> Repeat search left using previous *string* argument

makes more sense. It means,

Having typed <control-c>, a string and <control-h>, it is now possible to repeat the search for the next occurence of the same string on the line to the left of the cursor by depressing <control-c> followed immediately by <control-h>. *Edx* has memorized the string from your earlier search and it is not necessary to enter the string again in order to repeat the search.

The line from the *UNIX Manual* that reads...

<c> *number* <h> Move left *number* characters

means that if instead of typing a string after <control-c> you type a number, the cursor will move that number of characters to the left. Remember if you are looking for an actual number, say, "12" in the example below:

```
|135082 43 54544.54 336773 12 4555 4254 4423. 5 4020 845458 5 14 3 [3]4145
-
-
-
-
-
-
-
-
-
-
-
```

you must type <control-c>, next depress the space bar and then type the string "12". This will result in...

```
|135082 43 54544.54 336773 [1]2 4555 4254 4423. 5 4020 845458 5 14 3 34145
-
-
-
-
-
-
-
-
-
-
```

instead of...

```
|135082 43 54544.54 336773 12 4555 4254 4423. 5 4020 84[5]458 5 14 3 34145
-
-
-
-
-
-
-
-
-
-
```

In the second example the cursor moved twelve spaces (character positions or columns) to the left instead of finding the string "12".

More Commands from the UNIX Programmer's Manual

Notice that like <control-h> most commands can be made to do more things by preceding them with <control-c> or <control-c> followed by either a string or a number. Remember that <control-n> by itself means "move cursor to end of line." If we look in the *UNIX Manual* we find..

Beginning or End of Line

<n>	Move to the end of the line
<c> <n>	Move to the beginning of the line.
<c> *number* <n>	Move to column *number* of current line

Typing <control-c><control-n> moves the cursor to the beginning instead of the end of the line. Typing <control-c> followed by some number and then <control-n> will move the cursor to that many character-positions from the left-hand margin, that is, move the cursor to a specified horizontal position (column.)

The entry...

Top or Bottom Line

<g>	Move to the top of the screen.
<c> <g>	Move to the bottom of the screen.
<c> *number* <g>	Move to the line *number* on the screen

tells us that <control-c><control-g> is a fast way to move the cursor to the bottom of the screen, or, for that matter, to any line number of our choosing.

Of the SCREEN MOVEMENT COMMANDS, most of which we know already, all can be made more versatile by preceding them with a <control-c> or <control-c> and a number.

Of the TEXT MODIFICATION COMMANDS there are seven we want to comment on here.

The command <control-s> is similar to <control-d> in that it removes words to the left of the cursor instead of to the right.

The command <control-t> is essential for inserting lines into existing text. It opens up room above the line the cursor is located on.

As mentioned earlier, the command <control-u> is the standard mechanism for deleting an entire line. To remove a line of text, place the cursor on the line to be deleted and type <control-u>. To remove, say, three contiguous lines of text, place the cursor on the top-most of the three lines and type either three <control-u>'s in a row or <control-c> 3 <control-u>, as implied by the *UNIX Manual's*

<c> *number* <u>	Delete *number* lines starting with the current line

For now you should ignore where the *UNIX Manual* says

<c> <u>	Delete all characters between the cursor and the mark*

and

<c> *string* <u>	Delete one line and save it in the buffer named *string*
<c> *number string* <u>	Delete *number* lines and save it in the buffer named *string*

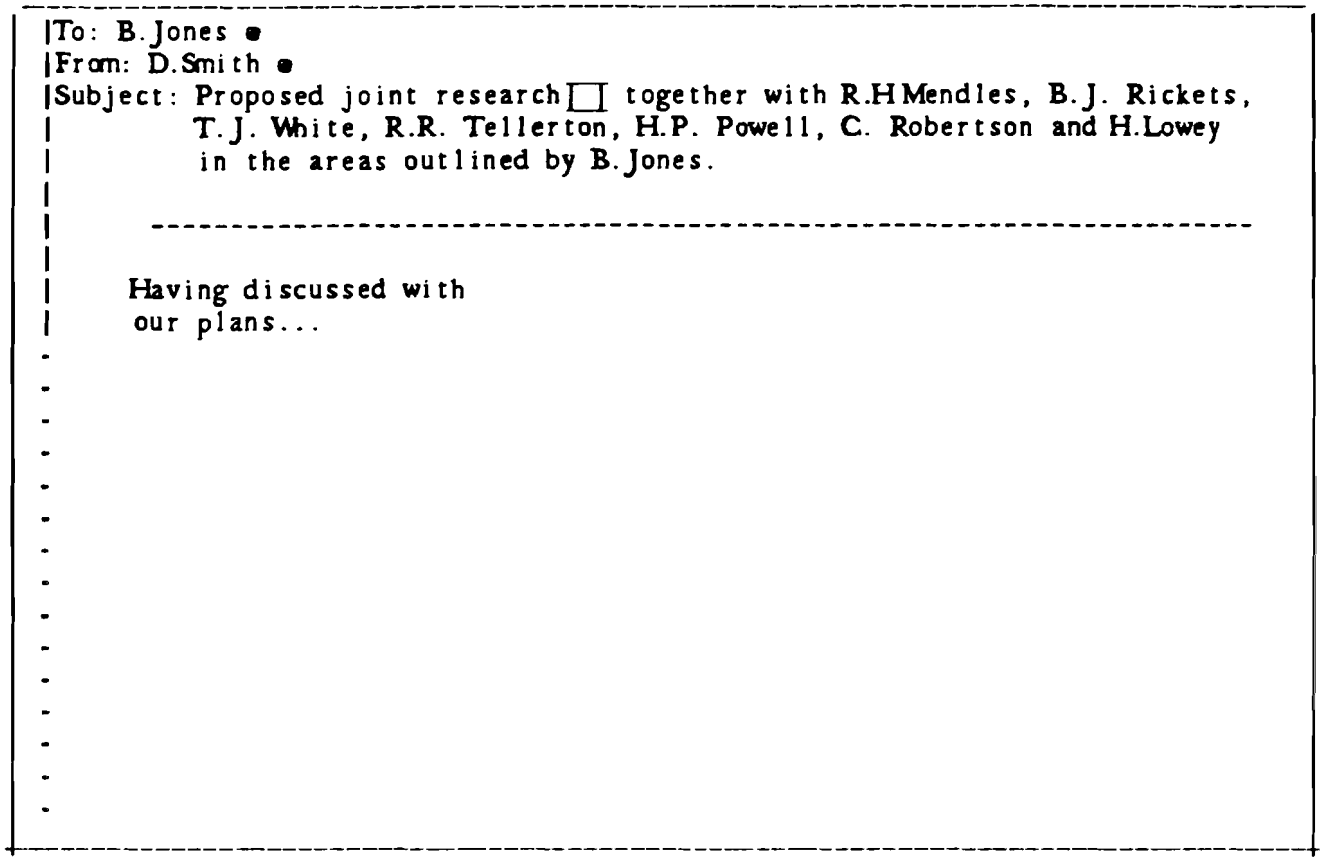
since these features have yet to be implemented. It is worthwhile knowing that typing <control-c> followed immediately by <control-x> will retrieve the last piece of text removed using <control-u>. This can be handy if you have inadvertantly removed something with <control-u> and want it back. We will talk more about <control-x> below.

Of considerable use, if you have a lot of cutting and pasting to do, are the commands <control-z> and <control-x>. To use these two commands you should first look at <control-z>.

Save Line

<z>	Save one line into buffer **a**
<c> *number* <z>	Save *number* lines in buffer **a**
<c> *string* <z>	Save the current line in the buffer named *string*
<c> *string number* <z>	Save the *number* lines in the buffer named *string*

Note that a "buffer" is a temporary storage area for the piece of text you want to save. There are currently ten buffers available in *edx* named "a", "b", "c", "d", "e", "f", "g", "h", "i", and "j". Where word *string* appears, you may use one of these ten lowercase letters to indicate the buffer you desire. If you type <control-z> and nothing else, the line the cursor is located on is stored in buffer "a" automatically. The quickest way to retrieve the line that you have stored in buffer "a" by using <control-z> is to simply type <control-x>. The text will be inserted above wherever the cursor is located. For saving more text in more buffers, use <control-z> and <control-x> together with the <control-c> command. For example, in the text below Smith will save two lines--the line beginning "Subject: Proposed..." and the one immediately below it--in the buffer "d" and then insert it above the line beginning "our plans...". The first step is to put the cursor on the topmost line of the text to be saved. The horizontal position of the cursor is irrelevant. In our example it is after the word "research".



Next Smith depresses <control-c>. The word "Arg: " appears in the upper left-hand corner of the screen together with the cursor.

```
|Arg: □
|From: D.Smith
|Subject: Proposed joint research together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|         in the areas outlined by B.Jones.
|
|      ------------------------------------------------------------------
|
|     Having discussed with
|     our plans...
-
-
-
-
-
-
-
-
```

and Smith next types "d" to specify buffer "d" and a "2" to specify how many lines are to be saved.

```
|Arg: d2□
|From: D.Smith
|Subject: Proposed joint research together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|         in the areas outlined by B.Jones.
|
|      ------------------------------------------------------------------
|
|     Having discussed with
|     our plans...
-
-
-
-
-
-
-
-
-
-
-
-
```

Smith next types <control-z> at which point the "Arg: d2" disappears and the screen looks the way it did before the <control-c> was depressed. Although nothing visably has changed, two lines have now been stored in buffer "d".

```
|To: B.Jones ●
|From: D.Smith ●
|Subject: Proposed joint research□ together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|         in the areas outlined by B.Jones.
|
|      ------------------------------------------------------------------
|
|     Having discussed with
|     our plans...
-
-
-
-
-
-
-
-
-
-
-
-
```

In order to insert these two lines above the line "our plans..." Smith moves the cursor to that line (again it is irrelevant where the cursor is horizontally.)

```
|To: B.Jones •
|From: D.Smith •
|Subject: Proposed joint research together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|         in the areas outlined by B.Jones.
|
|      ------------------------------------------------------------------
|
|     Having discussed with
|     [o]ur plans...
-
-
-
-
-
-
-
-
-
-
-
-
```

and types <control-c> again. And again the "Arg: " appears in the upper left hand corner, in reponse to which Smith types "d" (for buffer "d")

```
|Arg: d
|From: D.Smith
|Subject: Proposed joint research together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|         in the areas outlined by B.Jones.
|
|      ------------------------------------------------------------------
|
|     Having discussed with
|     our plans...
-
-
-
-
-
-
-
-
```

followed by <control-x> which results in the retrieval of the desired two lines.

```
|To: B.Jones
|From: D.Smith
|Subject: Proposed joint research together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|         in the areas outlined by B.Jones.
|
|      ------------------------------------------------------------------
|
|     Having discussed with
|Subject: Proposed joint research together with R.H Mendles, B.J. Rickets,
|         T.J. White, R.R. Tellerton, H.P. Powell, C. Robertson and H.Lowey
|     our plans...
-
-
-
-
-
-
-
-
```

Each time Smith types the sequence <control-c> d <control-x> the same two lines are inserted in front of the line the cursor is located on. This will continue until Smith quits *edx* for good or until the buffer "d" is filled with other text by means of the <control-c> *string number* <control-z> sequence.

The last two TEXT MODIFICATION COMMANDS that we suggest you master are <control-^> and <control-]>. These commands complement each other. The command <control-^> splits a line in two at the point the cursor is located at and <control-]> joins two lines--the one the cursor is on and the one immediately below it. These two commands are very useful for cutting and pasting and together with <control-z> and <control-x> give you great power to shuffle pieces of text around.

Finally, under miscellaneous, note that typing <control-@> (here we mean the character "@" and not a carriage return or newline) will restore a line to the way it looked before you made changes in it by means of other commands. It can be very disconcerting to type <control-@> after having spent the better part of a minute making some changes to a line.

As we mentioned in the beginning of this paper, you are automatically in "insert mode" when starting with *edx*. This means that when typing regular characters, text is inserted before any character the cursor is located on. By depressing first the <escape> key and then "o" you can put *edx* into "overwrite" mode. In this mode typing a character causes it to replace whatever character the cursor is located on. The overwrite mode can be useful for certain kinds of text replacement. It is very destructive, however, destroying whatever is in the path of the cursor. To go back to insert mode. just type <escape> followed by "i".

WARNINGS

At the present, *edx* on some terminals will introduce unwanted characters into text when a fast typist is using it. These characters can be removed like other typographical errors. For fast typists we recommend the program *ed* (see Kernighan.)

On some terminals the screen may become garbled as a result of attempting to do too many commands too quickly. To repaint the screen, that is, to remove the garbled text and see the screen as it should appear, type an <escape> followed by a lowercase "r". This <escape_r> may be typed at any time while using *edx*.

NROFF – HOW IT WORKS

Nroff is a text formatting program available on the IIASA computer. Our aim here is to give a brief description of how it works and introduce a few of its basic features. For more detail, see Using the Computer to Communicate: nroff/troff at IIASA for Formatting Documents* *(forthcoming IIASA working paper) and Documents for Use With the Phototypesetter: Version Seven (available from Computer Services.)*

To use the *nroff* features described in this paper you must first create an [input file] consisting of the text you want formatted and, embedded within the text, *nroff* codes (lines beginning with a period--British full stop) specifying how the text is to be formatted. Below is an example of such a file. Let us assume it was created by Smith using *edx* and that it is called "model".

```
.ND
.SH 1
Introduction
.PP
At the boundary of individual or social choice problems, resilience
is related to the ability or inability of decision makers to make
decisions in a consistent or systematic manner. Consider the
writings of Murphy in this
area.
His magnum opus,
.IT
Laws and Theories,
stands as a affirmation
of this fact accepted by the community as a whole
.PP
Why the tragedy of incompatibility of individual rationality
and social necessity or, in Murphy's terms,
of the disassociating of design and control complexity? Why the
breakdown in levels of understanding among control, design,
and resilience as a manifestation of the whole? Murphy gives insight
in to this when he writes:
.QU
When a person attempts a task, he or she will be
thwarted in that task by the unconscious intervention of some other presence
(animate or
inanimate).
Nevertheless, some tasks are completed, since the intervening presence
is itself attempting a task and, of course, subject to interference.
.PP
What are the corrollaries of this manifesting resilience of design
and management? What other features continue to impinge on the
```

Troff is a text formatting program like *nroff* that processes documents for an electrostatic printer-plotter or a phototypesetter. It can use the same input files as *nroff*. This working paper was formatted by *troff*.

equilibrium of a dis-equal, multi-faceted phase of unconscious
intervention...

It it from this input file that Smith, using *nroff*, creates an output file that looks as follows:

INTRODUCTION

At the boundary of individual or social choice problems, resilience is related to the ability or inability of decision makers to make decisions in a consistent or systematic manner. Consider the writings of Murphy in this area. His magnum opus, *Laws and Theories*, stands as a affirmation of this fact accepted by the community as a whole

Why the tragedy of incompatibility of individual rationality and social necessity or, in Murphy's terms, of the disassociating of design and control complexity? Why the breakdown in levels of understanding among control, design, and resilience as a manifestation of the whole? Murphy gives insight in to this when he writes:

> When a person attempts a task, he or she will be thwarted in that task by the unconscious intervention of some other presence (animate or inanimate). Nevertheless, some tasks are completed, since the intervening presence is itself attempting a task and, of course, subject to interference.

What are the corrollaries of this manifesting resilience of design and management? What other features continue to impinge on the equilibrium of a dis-equal, multi-faceted phase of unconscious intervention...

The *nroff* CODES in the input file tell *nroff*, in part, how to format the TEXT. In the example above, for example, the codes have the following meanings:

.ND NODATE (without this at the beginning of a file, *nroff* automatically appends the current date to the bottom of each page when formatting; something handy for drafts of a paper)

.SH 1 LEVEL ONE HEADER (text immediately following this code and preceding the next .PP code will be displayed as a level one header, that is, in uppercase--in *troff* in uppercase and in boldface; the sequence:

```
.SH 1
Introduction
.SH 2
Structural Overview
.PP
```

Text of paper here....

is also acceptable. There are three header levels available)

.PP INDENTED PARAGRAPH (text immediately following this code will begin a new, indented, paragraph--the code .LP will begin a new, left-adjusted paragraph)

.QU QUOTATION (following text is indented and with a shorter line length)

.IT ITALICIZE (the following line of text is underlined by *nroff*, italicized by *troff*)

To use the *nroff* program described in this paper, Smith types the command:

```
% run.nroff model ↵
```

and *nroff* reads from the input file "model" and automatically creates an output file called "model.pt" ("pt" stands for "printing" file) whose contents we saw above. Since it must be able to append the suffix ".pt" to the "printable" formatted output file it produces, the command *run.nroff* will not work with an input file whose name is longer than eleven characters.

The "model.pt" file produced by *nroff* is just like any other file. It can be printed on either a hardcopy or video terminal by typing

```
% sl model.pt ↵
```

It can be edited using *edx*. Note, however, that it is bad practice to edit output files. If you have additions or corrections to make to a file that has been "nroffed" you should modify the input file and then create a new output file by using *run.nroff*. To give a file a new name, use the "move command" [mv]. The command...

```
% mv model.pt globalmodel ↵
```

gives the file "model.pt" a new name.

If *rm's* first argument is a file name and its second argument is a directory pathname, the file will be moved to the indicated directory. Thus,

```
% mv jonesnote /mnt/smith/messages ↵
```

will move "jonesnote" to the directory messages.*

*The "copyfile" command [cp] is analogous to *mv* except that instead of changing a file's name or moving it to another directory, *cp* makes a copy of the original or puts a copy in the specified directory. For example,

Remember that

```
% rm model.pt ●
```

will destroy (remove) the file "model.pt".* Had an earlier version of "model.pt" existed in Smith's current directory when he or she typed

```
run.nroff  model ●
```

the earlier version of "model.pt" would have been destroyed.

In comparing the input and output files above, notice that the *nroff* program:

1) fills and hyphenates, that is, collects words (together with associated punctuation marks) from the input file until it can produce a line in the output file that is [right justified]. If necessary *nroff* hyphenates words and modifies the horizontal spacing between words to insure the lines in the output file end the same distance from the right-hand margin (except of course, if it is the last line in a paragraph that does not reach the right-hand margin;)

2) automatically sets left, right, top and bottom margins and paginates the document from page two on;

3) automatically single spaces the document unless told to do otherwise.

In the input file it is important not to hypenate words (divide them accross lines) since *nroff* does this automatically. If you hypenate a word, *nroff* will treat it as two words when filling lines in the output file, thus produc- ing strange look- ing words in your out- put file like these.

Start each line at the left-hand margin, that is, have no horizontal spaces at the beginning of a line. (It is also adviseable not to have any spaces at the end of a line as well.)

With these few facts it is possible to format simple documents on the IIASA computer. We suggest the reader experiment with his or her own files and see how the program works. As already mentioned, please see forthcoming papers in this series for more details.

```
% cp  jonesnote  joneslet
```

creates a file "joneslet" identical to "jonesnote". Similarly,

```
% cp  jonesnote  /mnt/smith/messages
```

puts a copy of "jonesnote" in the directory "messages". This new file also has the name "jonesnote". One must be careful using *mv* and *cp*. Moving or copying a file to a filename of an existing file will destroy the earlier file.

*It is a good idea to always remove unneeded files. Having many infrequently used files is also a bad practice. Call Computer Services at ext. 465 and have any file you won't be needing for a while put on magnetic tape for you. After this is done you can remove the file. Later when you need it, another phone call will get the file restored for you.

References

Computer Services, *Documents for Use With the Phototypesetter: Version Seven UNIX*

Kernighan, B.W., *A Tutorial Introduction to the UNIX Text Editor.*

Lathrop, C.L., *Using the Computer to Communicate: A Text Processing Workbook* (forthcoming IIASA working paper.)

Lathrop, C.L. and M.M.L.Pearson, *Using the Computer to Communicate: A Glossary for Users of IIASA's Computer* (forthcoming IIASA working paper.)

Lathrop, C.L. and M.M.L. Pearson, *Using the Computer to Communicate: nroff/troff* at IIASA for Formatting Documents (forthcoming IIASA working paper)

Pearson, M.M.L. (1980) *Using the Computer to Communicate: An Introduction to Computerized Conferencing* WP-80-72. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Pearson, M.M.L. and C.L. Lathrop (1980) *Using the Computer to Communicate: A User's Guide to Telecenter* WP-80-109. Laxenburg, Austria: International Institute for Applied Systems Analysis.

UNIX Time-Sharing System: UNIX Programmer's Manual, Seventh Edition, Volume 1, January, 1979.