Edit: A Tutorial

Ricki Blau

James Joyce

Computing Services
University of California
Berkeley, California 94720

ABSTRACT

This narrative introduction to the use of the text editor *edit* assumes no prior familiarity with computers or with text editing. Its aim is to lead the beginning UNIX† user through the fundamental steps of writing and revising a file of text. Edit, a version of the text editor *ex,* was designed to provide an informative environment for new and casual users.

We welcome comments and suggestions about this tutorial and the UNIX documentation in general.

September 1981

†UNIX is a trademark of Bell Laboratories.

Contents

Introduction

Text editing using a terminal connected to a computer allows you to create, modify, and print text easily. A *text editor* is a program that assists you as you create and modify text. The text editor you will learn here is named *edit.* Creating text using edit is as easy as typing it on an electric typewriter. Modifying text involves telling the text editor what you want to add, change, or delete. You can review your text by typing a command to print the file contents as they are currently. Another program (which we do not discuss in this document), a text formatter, rearranges your text for you into ‘‘finished form.’’

These lessons assume no prior familiarity with computers or with text editing. They consist of a series of text editing sessions which lead you through the fundamental steps of creating and revising text. After scanning each lesson and before beginning the next, you should try the examples at a terminal to get a feeling for the actual process of text editing. If you set aside some time for experimentation, you will soon become familiar with using the computer to write and modify text. In addition to the actual use of the text editor, other features of UNIX will be very important to your work. You can begin to learn about these other features by reading one of the other tutorials that provide a general introduction to the system. You will be ready to proceed with this lesson as soon as you are familiar with (1) your terminal and its special keys, (2) how to login, (3) and the ways of correcting typing errors. Let’s first define some terms:

program
: A set of instructions, given to the computer, describing the sequence of steps the computer performs in order to accomplish a specific task. The task must be specific, such as balancing your checkbook or editing your text. A general task, such as working for world peace, is something we can all do, but not something we can currently write programs to do.

UNIX
: UNIX is a special type of program, called an operating system, that supervises the machinery and all other programs comprising the total computer system.

edit
: *edit* is the name of the UNIX text editor you will be learning to use, and is a program that aids you in writing or revising text. Edit was designed for beginning users, and is a simplified version of an editor named *ex.*

file
: Each UNIX account is allotted space for the permanent storage of information, such as programs, data or text. A file is a logical unit of data, for example, an essay, a program, or a chapter from a book, which is stored on a computer system. Once you create a file, it is kept until you instruct the system to remove it. You may create a file during one UNIX session, end the session, and return to use it at a later time. Files contain anything you choose to write and store in them. The sizes of files vary to suit your needs; one file might hold only a single number, yet another might contain a very long document or program. The only way to save information from one session to the next is to store it in a file, which you will learn in Session 1.

filename
: Filenames are used to distinguish one file from another, serving the same purpose as the labels of manila folders in a file cabinet. In order to write or access information in a file, you use the name of that file in a UNIX command, and the system will automatically locate the file.

disk
: Files are stored on an input/output device called a disk, which looks something like a stack of phonograph records. Each surface is coated with a material similar to that on magnetic recording tape, and information is recorded on it.

buffer
: A temporary work space, made available to the user for the duration of a session of text editing and used for creating and modifying the text file. We can think of the buffer as a blackboard that is erased after each class, where each session with the editor is a class.

Session 1

Making contact with UNIX

To use the editor you must first make contact with the computer by logging in to UNIX. We'll quickly review the standard UNIX login procedure for the two ways you can make contact: on a terminal that is directly linked to the computer, or over a telephone line where the computer answers your call.

Directly-linked terminals

Turn on your terminal and press the RETURN key. You are now ready to login.

Dial-up terminals

If your terminal connects with the computer over a telephone line, turn on the terminal, dial the system access number, and, when you hear a high-pitched tone, place the telephone handset in the acoustic coupler, if you are using one. You are now ready to login.

Logging in

The message inviting you to login is:

login:

Type your login name, which identifies you to UNIX, on the same line as the login message, and press RETURN. If the terminal you are using has both upper and lower case, **be sure you enter your login name in lower case;** otherwise UNIX assumes your terminal has only upper case and will not recognize lower case letters you may type. UNIX types ‘‘login:’’ and you reply with your login name, for example ‘‘susan’’:

login: **susan** *(and press the RETURN key)*

(In the examples, input you would type appears in **bold face** to distinguish it from the responses from UNIX.)

UNIX will next respond with a request for a password as an additional precaution to prevent unauthorized people from using your account. The password will not appear when you type it, to prevent others from seeing it. The message is:

Password: *(type your password and press RETURN)*

If any of the information you gave during the login sequence was mistyped or incorrect, UNIX will respond with

Login incorrect.
login:

in which case you should start the login process anew. Assuming that you have successfully logged in, UNIX will print the message of the day and eventually will present you with a % at the beginning of a fresh line. The % is the UNIX prompt symbol which tells you that UNIX is ready to accept a command.

Asking for *edit*

You are ready to tell UNIX that you want to work with edit, the text editor. Now is a convenient time to choose a name for the file of text you are about to create. To begin your editing session, type **edit** followed by a space and then the filename you have selected; for example, ‘‘text’’. After that, press the RETURN key and wait for edit's response:

% **edit text** *(followed by a RETURN)*
"text" No such file or directory
:

If you typed the command correctly, you will now be in communication with edit. Edit has set aside a buffer for use as a temporary working space during your current editing session. Since ‘‘text’’ is a new file we are about to create the editor was unable to find that file, which it confirms by saying:

"text" No such file or directory

On the next line appears edit's prompt ‘‘:’’, announcing that you are in *command mode* and edit expects a command from you. You may now begin to create the new file.

The ‘‘Command not found’’ message

If you misspelled edit by typing, say, ‘‘editor’’, this might appear:

% **editor**
editor: Command not found
%

Your mistake in calling edit ‘‘editor’’ was treated by UNIX as a request for a program named ‘‘editor’’. Since there is no program named ‘‘editor’’, UNIX reported that the program was ‘‘not found’’. A new % indicates that UNIX is ready for another command, and you may then enter the correct command.

A summary

Your exchange with UNIX as you logged in and made contact with edit should look something like this:

login: **susan**
Password:
... A Message of General Interest ...
% **edit text**
"text" No such file or directory
:

Entering text

You may now begin entering text into the buffer. This is done by *appending* (or adding) text to whatever is currently in the buffer. Since there is nothing in the buffer at the moment, you are appending text to nothing; in effect, since you are adding text to nothing you are creating text. Most edit commands have two equivalent forms: a word that suggests what the command does, and a shorter abbreviation of that word. Many beginners find the full command names easier to remember at first, but once you are familiar with editing you may prefer to type the shorter abbreviations. The command to input text is ‘‘append’’. (It may be abbreviated ‘‘a’’.) Type **append** and press the RETURN key.

% **edit text**
: **append**

Messages from *edit*

If you make a mistake in entering a command and type something that edit does not recognize, edit will respond with a message intended to help you diagnose your error. For example, if you misspell the command to input text by typing, perhaps, ‘‘add’’ instead of ‘‘append’’ or ‘‘a’’, you will receive this message:

: **add**
add: Not an editor command
:

When you receive a diagnostic message, check what you typed in order to determine what part of your command confused edit. The message above means that edit was unable to recognize your mistyped command and, therefore, did not execute it. Instead, a new ‘‘:’’ appeared to let you know that edit is again ready to execute a command.

Text input mode

By giving the command ‘‘append’’ (or using the abbreviation ‘‘a’’), you entered *text input mode,* also known as *append mode.* When you enter text input mode, edit stops sending you a prompt. You will not receive any prompts or error messages while in text input mode. You can enter pretty much anything you want on the lines.

The lines are transmitted one by one to the buffer and held there during the editing session. You may append as much text as you want, and *when you wish to stop entering text lines you should type a period as the only character on the line and press the RETURN key.* When you type the period and press RETURN, you signal that you want to stop appending text, and edit responds by allowing you to exit text input mode and reenter command mode. Edit will again prompt you for a command by printing ‘‘:’’.

Leaving append mode does not destroy the text in the buffer. You have to leave append mode to do any of the other kinds of editing, such as changing, adding, or printing text. If you type a period as the first character and type any other character on the same line, edit will believe you want to remain in append mode and will not let you out. As this can be very frustrating, be sure to type **only** the period and the RETURN key.

This is a good place to learn an important lesson about computers and text: a blank space is a character as far as a computer is concerned. If you so much as type a period followed by a blank (that is, type a period and then the space bar on the keyboard), you will remain in append mode with the last line of text being:

.

Let’s say that you enter the lines (try to type **exactly** what you see, including ‘‘thiss’’):

This is some sample text.
And thiss is some more text.
Text editing is strange, but nice.
.

The last line is the period followed by a RETURN that gets you out of append mode.

Making corrections

If you have read a general introduction to UNIX, you will recall that it is possible to erase individual letters that you have typed. This is done by typing the designated erase character as many times as there are characters you want to erase.

The usual erase character varies from place to place and user to user. Often it is the backspace (control-H), so you can correct typing errors in the line you are typing by holding down the CTRL key and typing the ‘‘H’’ key. (Sometimes it is the DEL key.) If you type the erase character you will notice that the terminal backspaces in the line you are on. You can backspace over your error, and then type what you want to be the rest of the line.

If you make a bad start in a line and would like to begin again, you can either backspace to the beginning of the line or you can use the at-sign ‘‘@’’ to erase everything on the line:

Text edtiing is strange, but@
Text editing is strange, but nice.

When you type the at-sign (@), you erase the entire line typed so far and are given a fresh line to type on. You may immediately begin to retype the line. This, unfortunately, does not work after you type the line and press RETURN. To make corrections in lines that have been completed, it is necessary to use the editing commands covered in the next sessions.

Writing text to disk

You are now ready to edit the text. One common operation is to write the text to disk as a file for safekeeping after the session is over. This is the only way to save information from one session to the next, since the editor’s buffer is temporary and will last only until the end of the editing session. Learning how to write a file to disk is second in importance only to entering the text. To write the contents of the buffer to a disk file, use the command ‘‘write’’ (or its abbreviation ‘‘w’’):

:**write**

Edit will copy the contents of the buffer to a disk file. If the file does not yet exist, a new file will be created automatically and the presence of a ‘‘[New file]’’ will be noted. The newly-created file will be given the name specified when you entered the editor, in this case ‘‘text’’. To confirm that the disk file has been successfully written, edit will repeat the filename and give the number of lines and the total number of characters in the file. The buffer remains unchanged by the ‘‘write’’ command. All of the lines that were written to disk will still be in the buffer, should you want to modify or add to them.

Edit must have a name for the file to be written. If you forgot to indicate the name of the file when you began to edit, edit will print in response to your write command:

No current filename

If this happens, you can specify the filename in a new write command:

: **write text**

After the ‘‘write’’ (or ‘‘w’’), type a space and then the name of the file.

Signing off

We have done enough for this first lesson on using the UNIX text editor, and are ready to quit the session with edit. To do this we type ‘‘quit’’ (or ‘‘q’’) and press RETURN:

: **write**
"text" [New file] 3 lines, 90 characters
: **quit**
%

The % is from UNIX to tell you that your session with edit is over and you may command UNIX further. Since we want to end the entire session at the terminal, we also need to exit from UNIX. In response to the UNIX prompt of ‘‘ % ’’ type the command

% **logout**

This will end your session with UNIX, and will ready the terminal for the next user. It is always important to type **logout** at the end of a session to make absolutely sure no one could accidentally stumble into your abandoned session and thus gain access to your files, tempting even the most honest of souls.

This is the end of the first session on UNIX text editing.

Session 2

Login with UNIX as in the first session:

login: **susan** *(carriage return)*
Password: *(give password and carriage return)*
... A Message of General Interest ...
%

When you indicate you want to edit, you can specify the name of the file you worked on last time. This will start edit working, and it will fetch the contents of the file into the buffer, so that you can resume editing the same file. When edit has copied the file into the buffer, it will repeat its name and tell you the number of lines and characters it contains. Thus,

% edit text
"text" 3 lines, 90 characters
:

means you asked edit to fetch the file named ‘‘text’’ for editing, causing it to copy the 90 characters of text into the buffer. Edit awaits your further instructions, and indicates this by its prompt character, the colon (:). In this session, we will append more text to our file, print the contents of the buffer, and learn to change the text of a line.

Adding more text to the file

If you want to add more to the end of your text you may do so by using the append command to enter text input mode. When ‘‘append’’ is the first command of your editing session, the lines you enter are placed at the end of the buffer. Here we’ll use the abbreviation for the append command, ‘‘a’’:

: **a**
This is text added in Session 2.
It doesn’t mean much here, but
it does illustrate the editor.
.

You may recall that once you enter append mode using the ‘‘a’’ (or ‘‘append’’) command, you need to type a line containing only a period (.) to exit append mode.

Interrupt

Should you press the RUB key (sometimes labelled DELETE) while working with edit, it will send this message to you:

Interrupt
:

Any command that edit might be executing is terminated by rub or delete, causing edit to prompt you for a new command. If you are appending text at the time, you will exit from append mode and be expected to give another command. The line of text you were typing when the append command was interrupted will not be entered into the buffer.

Making corrections

If while typing the line you hit an incorrect key, recall that you may delete the incorrect character or cancel the entire line of input by erasing in the usual way. Refer either to the last few pages of Session 1 if you need to review the procedures for making a correction. The most important idea to remember is that erasing a character or cancelling a line must be done before you press the RETURN key.

Listing what's in the buffer (p)

Having appended text to what you wrote in Session 1, you might want to see all the lines in the buffer. To print the contents of the buffer, type the command:

: **1,$p**

The ‘‘1’’† stands for line 1 of the buffer, the ‘‘$’’ is a special symbol designating the last line of the buffer, and ‘‘p’’ (or **print**) is the command to print from line 1 to the end of the buffer. The command ‘‘1,$p’’ gives you:

This is some sample text.
And thiss is some more text.
Text editing is strange, but nice.
This is text added in Session 2.
It doesn't mean much here, but
it does illustrate the editor.

Occasionally, you may accidentally type a character that can't be printed, which can be done by striking a key while the CTRL key is pressed. In printing lines, edit uses a special notation to show the existence of non-printing characters. Suppose you had introduced the non-printing character ‘‘control-A’’ into the word ‘‘illustrate’’ by accidently pressing the CTRL key while typing ‘‘a’’. This can happen on many terminals because the CTRL key and the ‘‘A’’ key are beside each other. If your finger presses between the two keys, control-A results. When asked to print the contents of the buffer, edit would display

it does illustr^Ate the editor.

To represent the control-A, edit shows ‘‘^A’’. The sequence ‘‘^’’ followed by a capital letter stands for the one character entered by holding down the CTRL key and typing the letter which appears after the ‘‘^’’. We'll soon discuss the commands that can be used to correct this typing error.

In looking over the text we see that ‘‘this’’ is typed as ‘‘thiss’’ in the second line, a deliberate error so we can learn to make corrections. Let's correct the spelling.

Finding things in the buffer

In order to change something in the buffer we first need to find it. We can find ‘‘thiss’’ in the text we have entered by looking at a listing of the lines. Physically speaking, we search the lines of text looking for ‘‘thiss’’ and stop searching when we have found it. The way to tell edit to search for something is to type it inside slash marks:

: **/thiss/**

By typing **/thiss/** and pressing RETURN, you instruct edit to search for ‘‘thiss’’. If you ask edit to look for a pattern of characters which it cannot find in the buffer, it will respond ‘‘Pattern not found’’. When edit finds the characters ‘‘thiss’’, it will print the line of text for your inspection:

And thiss is some more text.

Edit is now positioned in the buffer at the line it just printed, ready to make a change in the line.

†The numeral ‘‘one’’ is the top left-most key, and should not be confused with the letter ‘‘el’’.

The current line

Edit keeps track of the line in the buffer where it is located at all times during an editing session. In general, the line that has been most recently printed, entered, or changed is the current location in the buffer. The editor is prepared to make changes at the current location in the buffer, unless you direct it to another location.

In particular, when you bring a file into the buffer, you will be located at the last line in the file, where the editor left off copying the lines from the file to the buffer. If your first editing command is ‘‘append’’, the lines you enter are added to the end of the file, after the current line — the last line in the file.

You can refer to your current location in the buffer by the symbol period (.) usually known by the name ‘‘dot’’. If you type ‘‘.’’ and carriage return you will be instructing edit to print the current line:

```
:.
And thiss is some more text.
```

If you want to know the number of the current line, you can type **.=** and press RETURN, and edit will respond with the line number:

```
:.=
2
```

If you type the number of any line and press RETURN, edit will position you at that line and print its contents:

```
:2
And thiss is some more text.
```

You should experiment with these commands to gain experience in using them to make changes.

Numbering lines (nu)

The **number (nu)** command is similar to print, giving both the number and the text of each printed line. To see the number and the text of the current line type

```
:nu
      2   And thiss is some more text.
```

Note that the shortest abbreviation for the number command is ‘‘nu’’ (and not ‘‘n’’, which is used for a different command). You may specify a range of lines to be listed by the number command in the same way that lines are specified for print. For example, **1,$nu** lists all lines in the buffer with their corresponding line numbers.

Substitute command (s)

Now that you have found the misspelled word, you can change it from ‘‘thiss’’ to ‘‘this’’. As far as edit is concerned, changing things is a matter of substituting one thing for another. As *a* stood for *append,* so *s* stands for *substitute.* We will use the abbreviation ‘‘s’’ to reduce the chance of mistyping the substitute command. This command will instruct edit to make the change:

2s/thiss/this/

We first indicate the line to be changed, line 2, and then type an ‘‘s’’ to indicate we want edit to make a substitution. Inside the first set of slashes are the characters that we want to change, followed by the characters to replace them, and then a closing slash mark. To summarize:

2s/ *what is to be changed* / *what to change it to* /

If edit finds an exact match of the characters to be changed it will make the change **only** in the first occurrence of the characters. If it does not find the characters to be changed, it will respond:

```
Substitute pattern match failed
```

indicating that your instructions could not be carried out. When edit does find the characters that you want to change, it will make the substitution and automatically print the changed line, so that you can check that the correct substitution was made. In the example,

: **2s/thiss/this/**
And this is some more text.

line 2 (and line 2 only) will be searched for the characters ‘‘thiss’’, and when the first exact match is found, ‘‘thiss’’ will be changed to ‘‘this’’. Strictly speaking, it was not necessary above to specify the number of the line to be changed. In

: **s/thiss/this/**

edit will assume that we mean to change the line where we are currently located (‘‘.’’). In this case, the command without a line number would have produced the same result because we were already located at the line we wished to change.

For another illustration of the substitute command, let us choose the line:

Text editing is strange, but nice.

You can make this line a bit more positive by taking out the characters ‘‘strange, but ’’ so the line reads:

Text editing is nice.

A command that will first position edit at the desired line and then make the substitution is:

: **/strange/s/strange, but //**

What we have done here is combine our search with our substitution. Such combinations are perfectly legal, and speed up editing quite a bit once you get used to them. That is, you do not necessarily have to use line numbers to identify a line to edit. Instead, you may identify the line you want to change by asking edit to search for a specified pattern of letters that occurs in that line. The parts of the above command are: