UnixWorld ‘‘New To Unix’’ Column: September 1988

Creating and Editing Files

By Augie Hansen

Much of the time and energy you spend working on any computer system involves creating and editing text tiles. Office workers use editors and word processors to create the basics that make a business work: memos, letters, and dreaded meeting notices.

Programmers use editors to create specification and design documents, program source files, and, if encouraged or threatened, support program documentation. Virtually everyone in a computer-equipped organization can benefit from electronic mail, which also involves editing.

You need to master the fundamentals of editing early in your training because almost everything else you will learn depends to some extent on your ability to create and maintain text files.

In a UNIX environment, a text file is a disk file that contains only ordinary characters: letters, digits, punctuation marks, and a few control characters. These characters are represented within a computer as numeric codes that are defined by the American Standard Code for Information Interchange (ASCII) or by some other codification scheme. You create text files by using a text editor.

A text editor is not a word processor. A word processing program combines two tasks-text editing and text processing-into one program. Several word processing programs are available for UNIX systems, but none is provided as a standard component of a UNIX system.

The UNIX system offers a ‘‘divide and conquer’’ approach to doing document preparation. In this paradigm, you use an editing tool to create files that contain both text and formatting codes. The codes are interpreted by a separate text processing program, such as Nroff (pronounced en-roff), which applies the requested character, paragraph, and other formatting to printed and displayed output.

Most word processing programs use special non-ASCII codes to represent formatting information, so they are usually unsuitable for preparing standard text files. Some word processors provide a way for you to save files in pure text form, and they can be used as text editors.

Visual Editing

You should learn to use at least one of the supported UNIX editors. Your choices are Ed, Ex, and Vi, which are all available on most UNIX systems.

Last month we looked briefly at the line-oriented editor Ed and the process of creating and saving a file. This month we will explore the primary editing features of Vi, the officially supported UNIX full-screen visual editor. We mention the popular public-domain Emacs editor briefly at the end.

First you will learn about the initial setup required to use Vi and how to start the editor from the UNIX command line. Then you will learn how to enter text, how to perform the basics of selecting and editing text, and how to save your work and quit editing.

The following material tells you the minimum amount of information you would know to start using the Vi editor. When you have become reasonably proficient in using this subset of visual editing features, you can investigate other Vi features that will help you do more complex editing tasks. You can also learn some of the many shortcuts that will help you work more efficiently.

Initial Setup

Being a visual editor, Vi must learn how to work with your terminal. The UNIX system stores information needed by the shell and application programs in environment variables. One of those environment variables is `TERM`, which should be set to indicate the type of terminal you are using.

Your system administrator may have already set the `TERM` variable for you. To find out, type ‘‘echo $TERM<CR>’’ and check the output.

Note that the angle brackets surrounding the letters ‘‘CR’’ indicate that the letters represent a special key on the keyboard rather than a sequence of individual letters. In particular <> means press the carriage return key. Later we will use <> to represent the escape key.

If `TERM` is not defined, you will see a blank line displayed. If it is defined, you will see the terminal identification string on your screen. For a DEC VT100 terminal, one of the very popular terminal types, the ID would be vt 100. There are hundreds of different terminal types in use today. Table 1 shows a few of the more popular terminals and their ID strings.

If `TERM` is not defined, use the following commands to tell Vi about your terminal. Let’s say you have a Televideo 950 terminal. If you have the UNIX System V Bourne shell, type ‘‘TERM=tvi950<CR>’’ followed by ‘‘export TERM<CR>’’. If you have the Berkeley C shell version, type ‘‘setenv TERM tvi950<CR>’’.

Look up the ID for your terminal in the user manuals or get the required information from your system administrator. The purpose of the `export` command is to tell the UNIX shell to pass copy of the `TERM` variable to programs such as Vi when they start running. The C shell `setenv` command assigns the value and exports it in a single operation, so you don’t need a separate `export` command.

To avoid having to key in the `TERM` setting and exporting commands each time you log in, add the commands to to your Bourne or Korn `.profile` or C shell `.login` initialization file file. You’ll be able to do that after you learn to use a text editor. But be aware that if you log in from a different terminal from time to time, you will need to override the `TERM` value manually as shown above.

Starting the Editor

To start the Vi editor, simply type its name as a command. You can also provide a file name or a list of file names as arguments. Let’s create a file called `quotes.txt` and place a famous quotation in it. After your shell prompt, type ‘‘vi quotes.txt<CR>’’.

This command starts the Vi editor and attempts to read in the file `quotes.txt`.. If the file exists, it is read into the editing buffer and displayed. If the file does not exist, Vi displays the following message on the last screen line: `"quotes.txt"[New file]`. The screen fills with tildes ()~down the left side of the screen to indicate that there is no text in the file.

The editing buffer is an area in the computer's memory where your text input and editing are maintained. All text input and editing actions affect the editing buffer in some way. At any time, all or portion of the editing buffer may be displayed on your terminal screen.

The current position in the editing buffer is indicateed by the cursor. Text input and editing operations usually take place at or near the cursor position. The Vi editor usually keeps the cursor in view in the editing window on your screen. Some commands, such as global search and replace, have effects that may not be visible on the screen.

Creating Files

You can append text by using the append (`a`) command, which has the general form: **`a`** ***`new text`*** **`<ESC>`**. Pressing the Escape key ends the text-input operation. The new text is placed in the editing buffer after (to the right of) the cursor position.

Let's append a quote from Mark Twain to the file `quotes.txt`), which is currently empty. Note that you're actually appending text to a buffer in volatile memory. The buffer contents will be saved to a permanent file named `quotes.txt` on disk in a later step. First type the append commend: `a`. You do not press a carriage return key following visual-mode commands. The command specified by the command letter takes effect immediately. Next type the text to be appended to the file:

```
"Work consists of whatever a
body is obliged to do...
Play consists of whatever a
body is not obliged to do."
              Mark Twain
```

End the text-input operation by pressing the Escape key. Following the append command, the cursor rests on the last text character that you typed into the editing buffer. Listing 1A shows how the screen looks after append operation.

Another command that puts text in the editing buffer is the insert (`i`) command. It works like append, except that the new text is inserted ahead of the cursor position. The form of an insert command is as follows: **`i`*****`text`*****`<ESC>`**.

Now let's insert a set of title lines at the beginning of the buffer. To do so, we must move the cursor, which is now on line 6, back to the first like in the buffer.

To move quickly to any line in the editing buffer, type the line number followed by the ''goto'' (`G`) command. Thus, to move to line number one, type ''1G'', and the cursor will move to the beginning of the first line. The beginning of a line in this context is the first column in which a visible character is displayed.

Table 2 contains a summary of the primary cursor-positioning commands. Each of the commands can take a preceding count that effectively multiplies the effective range of the action. For example, typing ‘‘5j’’ moves the cursor down five lines in the editing buffer, scrolling the window if necessary to keep the cursor in view. If there are not enough characters or lines to satisfy the request, the action is ignored, and the terminal bell sounds.

If your terminal has arrow keys, you can probably use them to move the cursor by column and lines. However, some terminal (HP 2621, for example) require that you press the Shift key at the same time as the function keys to get the required effect. Many video terminals don’t have function keys or arrow keys. Now type the insert command, `i`, and then type the text to be inserted:

```
FAMOUS QUOTATIONS
-----------------
```

Terminate the insert operation by pressing the Escape key. The insert operation puts three additional lines in the file ahead of the previously entered text. Following the insert operation, the screen appears as shown in Listing 1B.

Saving Text in a File

After you have inserted or appended text to the editing buffer, you should save it to disk by using a write (`w`) command (unless you don’t want to preserve the changes). Now type ‘‘:w<CR>’’ to write the contents of the editing buffer to the remembered file name. When you type the colon, Vi moves the cursor to the last line on the screen and prompts you with a colon and waits for a command.

For this example, the command is the write command, to which Vi responds: `"quotes.txt" 9 lines, 178 characters`. This tells you that it saved the file successfully. Now the contents of the editing buffer and the disk file are identical.

You can also save the editing buffer to a different file by using a command of the form **`:w`** ***`file`*****`<CR>`**. In this case, *file* is a name other than that of the current file. By specifying a range of lines to the write command, you can save a portion of the editing buffer to a file. For example, typing ‘‘:4, 9w twain<CR>’’ saves a copy of the Twain quotation without the header lines in a file name `twain`. The contents of the buffer and the `quotes.txt` file are not affected by this command.

Editing Text

The Vi editor is designed around a simple concept. You issue editing commands that specify an action and an object upon which to perform the action. Table 3A lists some of the more important actions, and Table 3B lists some of the more frequently used objects.

Forming commands from the actions and objects, you can delete, change, and substitute text easily. To delete the current word, type the command `dw` and use the command `3dW` to delete the current word and the next two while ignoring embedded punctuation that would normally delimit words. To delete from the cursor to the end of the current sentence, type ‘‘d)’’. The closing parenthesis can be thought of as the lens of an eyeball looking to the right in the file.

Doubling the action letter causes the action to affect entire lines. Thus `dd` delete the current line, and `5dd` deletes the current line and the next four lines.

Experiment on a file to see how the various actions and objects work together to let you edit text. Don't forget to use repetition counts to multiply the effects of commands.

One last command and we'll call it a month. To quit the editor, use the quit (`q`) command and type: ``:q<CR>''. If you have made any changes to the buffer since the last time you saved the file, Vi prints a diagnostic message ``No write since last change (:quit! overrides).''. If you really want to quit without saving changes, you can type a variation of the quit command: ``:q!<CR>''. This does an immediate exit without checking for unsaved changes.

Although this month's look at Vi has covered only a small percentage of the editor's commands, it should be sufficient to get you started. In future installments of this column, I will touch on other aspects of visual and line-oriented editing in the context of other UNIX topics.

Edited by Becca Thomas / Online Editor / UnixWorld Online / editor@unixworld.com

Go To Content   Search

Text entry and HTML markup by Arlene Lee
Last Modified: Sunday, 10-Mar-96 10:42:02 PST