UnixWorld Online: ‘‘Answers to Unix’’ Column: No. 002

Questions regarding this column should be sent to the author at ray@cse.ucsc.edu.

VInsight, Update 2000, Permissions on Open Files

By Ray Swartz

I’ve Something In My `vi`

Question: I switched from a mainframe to a Unix system four years ago and have gradually come to feel comfortable in my new environment. However, there are two mainframe editor features that I have not yet learned how to perform with `vi`. Can they be done?

The first feature is excluding lines. With the mainframe editor, I could look at and operate on separate portions of a file without splitting the screen and without bothering with line numbers simply by excluding a block of lines from the display.

The second feature was columnwise operation. I’d like to be able to operate on a set of characters only if they occur in certain columns of the text. I know I can do this with `awk` or `sed`, but I’d like to be able to do so with `vi`.

Steve Palmer / Norristown, Pennsylvania

Answer: While I have been using `vi` for at least 10 years now, there are still many things I don’t know (nor care to know :-) about it. Thus, to get a definitive answer to your question, I contacted Walter Zintz, UnixWorld Online’s Book Review Editor and frequent contributor. Walter knows just about all there is to know about `vi`. He provided invaluable help in preparing this answer:

The way to exclude lines from an edit session is to temporarily delete them from the file. Then, when you are done with your edits, return them to their original location.

The temporary deletes are done by ‘‘deleting into a buffer.’’ The `vi` editor provides buffers identified by single letters where you can store text during an editing session. To delete a block of lines into a named buffer, precede your deletion command with a double quote then the letter designation, as in `"a`, finally suffix `+0`. For instance, to delete five lines into buffer ‘‘a’’ enter: `"a5dd`. The `+0` suffix grabs the entire line at the start and end of the block.

Once you’ve deleted the text into a buffer, mark the line just above the deletion with the same letter--by typing ‘‘m’’ followed by the letter--so you will remember where the deleted text should reappear. Then, when you’ve finished your edits, just return to the marked line--by typing a single quote then the letter--and restore the deleted text block by typing `"ap`.

Incidentally, the deleted text block is safe inside a letter-named buffer, even switching to edit another

file won't lose it. Of course, you'll lose the buffer contents if you exit the editor.

Columnwise editing can be done in several ways. To move to, say, the 27th character position type the digits ''27'' followed by a vertical bar while in visual-command mode. This notation will also work as an address for editing commands; that is, typing ''d27|'' will delete the characters from the cursor position to that 27th character position, whether the direction be forward or backward.

To replace ''25'' with ''39'' if and only if the digits ''25'' are in the 7th and 8th character positions on the line, type:

```
s/^\(.....\)25/\139
```

followed by a return. The six dots are metacharacters matching any six characters at the beginning of the line, and the `\1` in the replacement pattern tells `vi` to insert whatever was found inside the first `\( \)` bracketed regular expression sequence, that is, the actual characters found, not the six dots.

If you know exactly how many characters are in the line, you can save the trouble and hazard of typing long strings of dots when you are working on columns near the right-hand edge. To make the above substitution in columns 55 and 56 of a 60 column line, type the following then a return:

```
s/25\(....\)$/39\1
```

To edit columns of figures or text found in tables, then use ''ex'' mode ''depends.'' To change ''yes'' to ''no'' only in the third column, (using spaces to separate columns) type:

```
s/^\( *[^ ][^ ]* *[^ ][^ ]* *\)yes/\1 no
```

This only works if there are no space characters within any column. (The space before the ''no'' keeps the change from moving the subsequent column's entries one space to the left.)

If there may be one or more single spaces within a column (for instance, the column contains phrases with spaces between the words), and there are always at least two space characters separating one column from another horizontally, then it's only necessary to change the single spaces (within column entries) to some other character before editing. A command such as:

```
s/\([^ ]\) \([^ ]\)/\1+\2/g/
```

will do this. Be sure to choose a substituted character that does not appear as itself anywhere in the text your editing. Next, do your editing as in my previous comments, then change back all the space-substitute characters to spaces.

If the edits you are performing are repetitive, you may wish to use macros instead of inventing and typing a new fancy command for each set of changes. Macros can be defined with the editor's `map` command. For instance, to define a macro named `Q` to be the command to delete five lines, enter from visual-command mode:

```
:map Q 5dd
```

From that time forward, whenever you type a capital ''Q'' in visual-command mode you delete five lines beginning with the line where the cursor is positioned. Of course, you can still enter a capital ''Q''

while entering text because that's performed in text-entry mode, not visual-command mode. To make a macro definition permanent across subsequent invocations of the editor, you'll need to place its definition in the `vi` initialization file, `.exrc`.

Another convenient way to work with text is to mark it with the `m` command. Type ''m'' followed by a letter to specify a mark. So typing ''ma'' marks the current line as ''a.'' There are two ways to return to the ''a'' line while in visual-command mode: use the `'a` command to return to the first non-whitespace character, but use `` `a `` to bring the cursor to the same character the cursor was positioned when the line was marked, anywhere on the line.

Dates and Confused

*Feedback:*Your remarks on the year 2000 in a previous column (May, 1994 issue of *Open Computing*) inspired me to check the behavior of some programs with the clock set to 23:59 hours--a minute prior to midnight--on February 28, 2000.

There is quite a lot of disagreement in the literature that the year 2000 is in fact a leap year. Some sources stick to the 4/100/400 rule, others add the ''divisible by 2000'' as yet another exception to the list of exceptions to the rule.

Of course, the software also disagrees on this issue (it was written by humans after all). An interesting inconsistency can be found among SCO Unix software modules.

On SCO, the `cal` utility thinks 2000 is a leap year (as you showed in your column). On the other hand, `date` doesn't think so. I set the clock to Feb 28, 2000 23:59 hours, waited a minute and typed ''date'' a second time: `date` reported that it's now March 1!

I'm definitely taking the day after Feb 28, 2000 off!

Kees Hendrikse / Enschede, The Netherlands

The More Things Change, The More They Stay The Same

Question: When I change the permissions of my terminal (`chmod 000 `tty`) I had thought that I would not be able to write to my screen because I no longer have write permission. Not so! Even though the `ls -l` command output shows no access permission, I can type (read the keyboard) and the characters are echoed (written) to the screen. So why can I type characters on the screen even though I don't have permission?

P.Ramadurai / Bangalore, India

Answer: It appears we have a paradox here. How can you write to a device after you remove its write permission? Yet, everything makes sense, once you understand how open files are divorced from their disk-based counterpart.

File permissions are checked only when a process tries to open the disk-based file. Read permission is checked when you open for input, write permission when you open for output or append. These permissions work the same for your terminal because it's accessed through a disk-based (device) file.

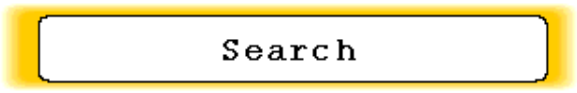
After a file has been opened in the requested mode, the connection between the process and the file is noted by the process-table entry in the kernel. In fact, the file descriptor returned by the `open()` system call is an index into an array of open-file descriptors in the process-table entry for the process that opened the file. Most important for our discussion: the permissions of the disk-based file aren't consulted again.

The terminal file was opened by the program that monitors the terminal port for a log-in request, generally by `getty`, `ttymon`, or possibly some other terminal-monitoring program. This open file is inherited by your log-in shell. Any changes to the access permissions on your already-opened terminal will have no effect on your ability to read from or write to it.

Your question reminds me of a related phenomenon: creating an ''invisible file'' that no one else can list--with `ls`--much less access. You open a file then erase it from the file system. You can still access the file because the program that opened the file refers to it through the file descriptor, not the file name on disk. This approach is frequently used for temporary files that you don't want accessed or overwritten by other processes. I've written a demonstration program to illustrate:

```
#include <stdio.h>

main()
{
   FILE *out;                    /* pointer to file */
   char str[100];                /* buffer */

   out = fopen("hidden", "w+"); /* open for appending */

   printf("Before file 'hidden' removed:\n\n");

   system("ls -C");              /* check that file is there */

   system("rm hidden");          /* remove file from directory */

   printf("\nAfter file 'hidden' removed:\n\n");

   system("ls -C");              /* verify file is gone */

   /* write into file */
   fprintf(out, "This is a line written to an invisible file\n");

   fseek(out, 0L, 0);            /* move to beginning of file */

   fgets(str, 100, out);         /* read line just written to file */

   printf("\n%s", str);          /* print out line just read */

   exit(0);                      /* file deallocated upon exit */
}
```

Copyright © 1995 The McGraw-Hill Companies, Inc. All Rights Reserved.
Edited by Becca Thomas / Online Editor / UnixWorld Online / beccat@wcmh.com



Last Modified: Sunday, 10-Sep-95 09:16:59 PDT