Speaking UNIX: The new and improved Vim editor

Edit your code on virtually any platform

Skill Level: Intermediate

Adam T. Cormany (acormany@yahoo.com)
National Data Center Manager
Scientific Games Corporation

19 Aug 2008

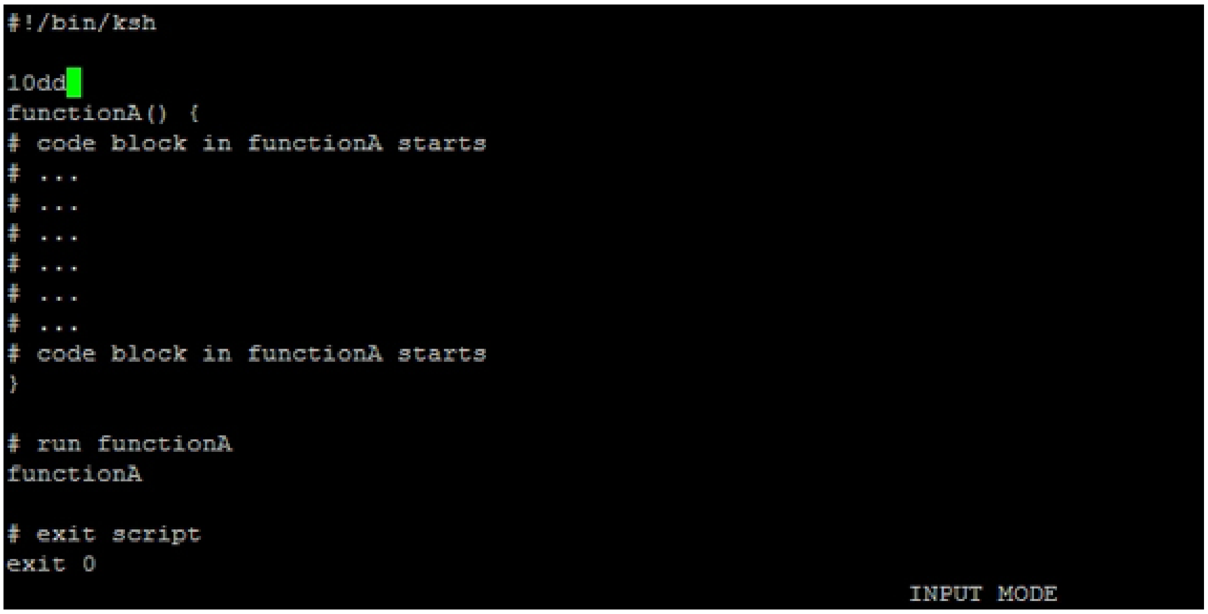
If you've worked on IBM® AIX®, another flavor of UNIX®, or Linux®, you've more than likely used the vi editor. Since its conception in 1976, vi has become a staple for anyone wanting to edit files. How could someone make a more powerful editing tool than vi, you may ask? The answer is Vim, and this article provides details on the many enhancements that have made Vim a highly used and acceptable editor in the world of UNIX and Linux.

The vi program is a powerful text editor. William Joy, co-founder of Sun Microsystems, originally wrote the vi editor in 1976 for an early version of Berkeley Software Distribution (BSD) UNIX. Rumor has it that Bill wrote vi in a weekend, but he says the claim is untrue. The program was named after the `visual` command in the extended line editor for UNIX—*ex,* for short.

The vi editor is an extremely powerful editor, with several features that many don't know even exist. Vi is a *modal editor,* meaning that the program produces different results when other settings are placed on the program. There are three distinct modes in vi: *command, insert* (or *input*), and *line.* When operating in insert mode, text is written to a temporary file being edited; while in normal mode, the same keystrokes provoke commands embedded in the editor. To enter insert mode, simply press the I key; to exit to command mode, press Escape. (I explain line mode in further detail later in this article.)

For example, in insert mode, if a you type the string `10dd`, that string would be written to the temporary file, as shown in Figure 1.

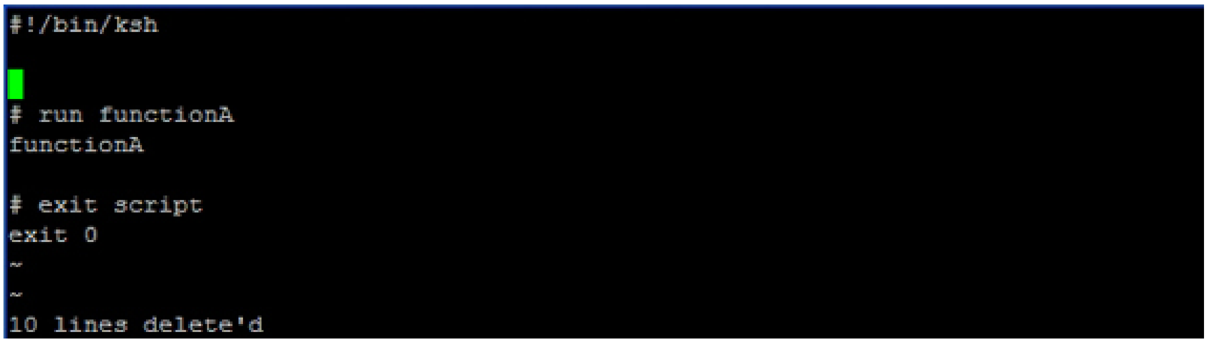
Figure 1. Typing dd in insert mode



```
#!/bin/ksh

10dd
functionA() {
# code block in functionA starts
# ...
# ...
# ...
# ...
# ...
# ...
# code block in functionA starts
}

# run functionA
functionA

# exit script
exit 0
                                                                  INPUT MODE
```

However, if you were in command mode, the string `10dd` would delete the 10 lines from the temporary file starting from where the cursor is currently placed, as shown in Figure 2.

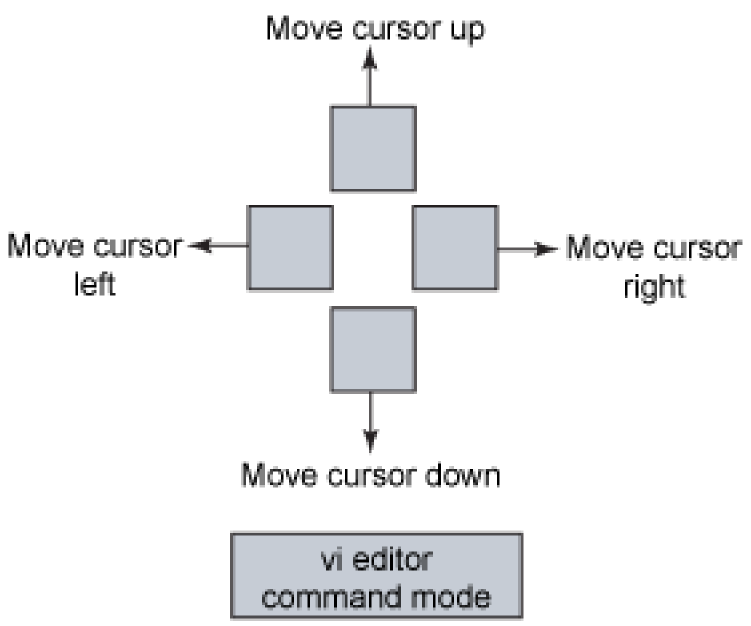
Figure 2. Typing dd in command mode



```
#!/bin/ksh


# run functionA
functionA

# exit script
exit 0
~
~
10 lines delete'd
```

Another example is cursor movement. In command mode, the keys H, J, K, and L move the cursor left, down, up, and right, respectively, as shown in Figure 3. In insert mode, these letters are displayed, instead.

Figure 3. Cursor movement in vi

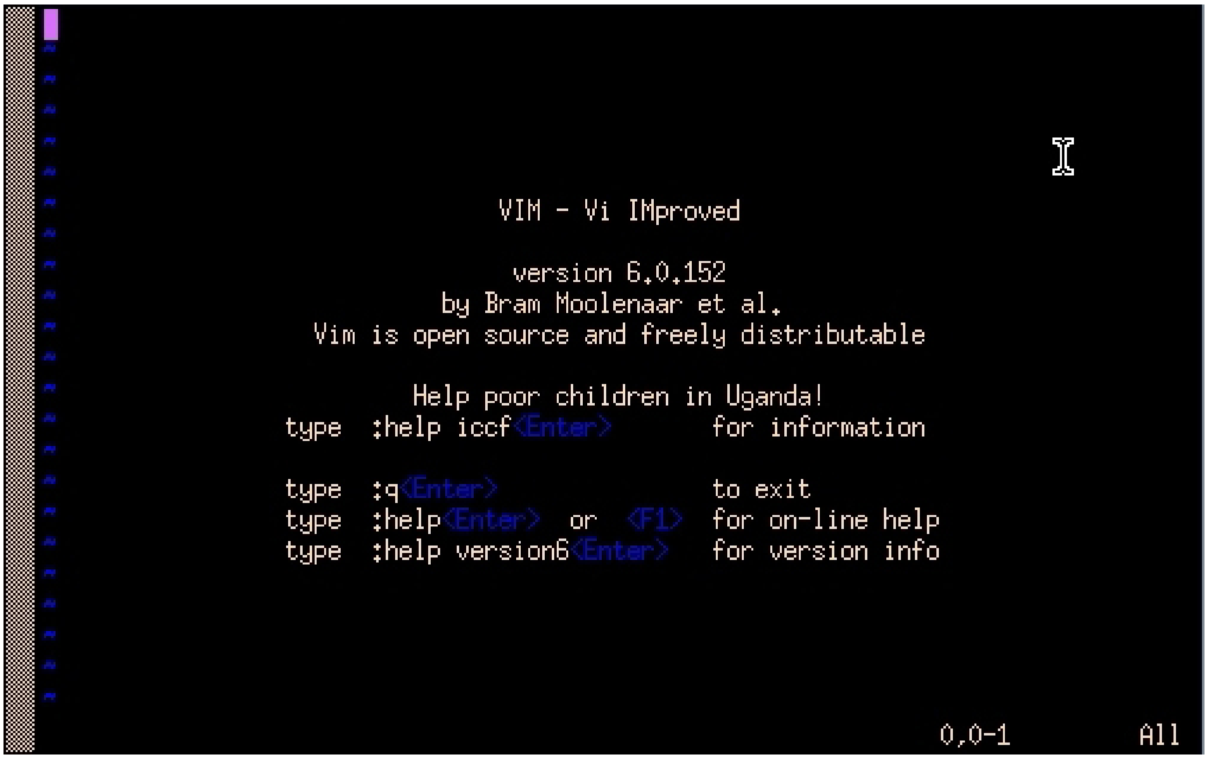


Typically, UNIX users either use vi or another editor called Editor Macros (Emacs) written by Richard Stallman in 1976. Many choose vi, however, because it's lightweight, starts faster, and uses less memory.

What is Vim?

Vim, or *Vi Improved*, is an extended version of vi written by Bram Moolenaar in 1991. The editor was originally designed for the Amiga computer but soon spread through UNIX in 1992. Like vi, Vim is based on command mode and insert mode as a text user interface (TUI)—shown in Figure 4.

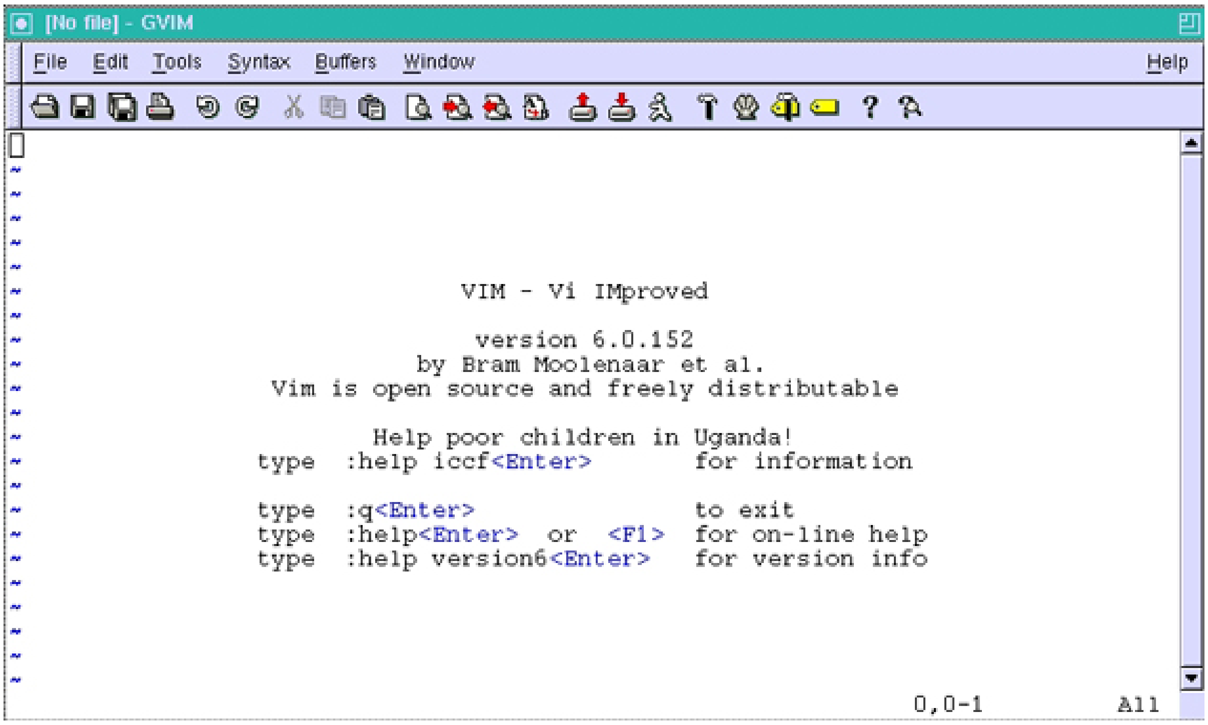
Figure 4. The Vim TUI



```
                     VIM - Vi IMproved

                     version 6.0.152
                 by Bram Moolenaar et al.
        Vim is open source and freely distributable

                 Help poor children in Uganda!
type  :help iccf<Enter>        for information

type  :q<Enter>                to exit
type  :help<Enter>  or  <F1>   for on-line help
type  :help version6<Enter>    for version info

                                              0,0-1          All
```

However, it does offer a graphical user interface (GUI) appropriately named *gVim*—shown in Figure 5.

Figure 5. The Vim GUI



```
                     VIM - Vi IMproved

                     version 6.0.152
                 by Bram Moolenaar et al.
        Vim is open source and freely distributable

                 Help poor children in Uganda!
type  :help iccf<Enter>        for information

type  :q<Enter>                to exit
type  :help<Enter>  or  <F1>   for on-line help
type  :help version6<Enter>    for version info

                                              0,0-1          All
```

Vim commands

Internal commands within Vim are similar to those within the vi editor. Table 1 provides the cursor movement commands within Vim.

Table 1. Vim commands for effecting cursor movement

Command	Action
h	Move cursor left
j, Plus Sign (+), Enter, or Return	Move cursor down
k, Minus Sign (-)	Move cursor up
l	Move cursor right
}	Move cursor to the end of the current paragraph
{	Move cursor to the beginning of the current paragraph
)	Move cursor to the end of the current sentence
(	Move cursor to the beginning of the current sentence
^	Move to the first non-blank character in the current line
$	Move to the end of the current line
0 (zero)	Move to the beginning of the current line
w or W	Move to the beginning of the next word
b or B	Move to the beginning of the previous word
e	Move to the end of the next word
H	Move to the first line of the screen
M	Move to the middle line of the screen
L	Move to the last line of the screen
G	Move to the end of the file
gg	Move to the beginning of the file
:n	Move to line *n*

Table 2 shows the Vim commands for deleting text.

Table 2. Vim commands for deleting text

Command	Action
d	Delete region selected
dd	Delete the entire current line
10dd	Delete 10 lines, starting with the current line
dw	Delete words from the current position onward
db	Delete words from the left of the current cursor position backwards
dl	Delete the character at the current cursor position
dh	Delete the character to the left of the current cursor position
d0 (zero)	Delete text from the current cursor position to the beginning of the line
D | d$	Delete the entire line starting at the current cursor position
x	Delete the character at the cursor's current position
X	Delete the character before the cursor's current position

Table 3 provides several other useful Vim commands.

Table 3. Common Vim commands

Command	Action
ih	Enter insert mode and insert at the current cursor position
I	Enter insert mode and insert at the beginning of the current line
a	Enter insert mode and append after the cursor
A	Enter insert mode and append to the end of the current line
c	Change the region selected
C	Change the entire line starting at the cursor's current position
o	Create a new blank line beneath the line in which the cursor is currently located and move the cursor to the beginning of the new

	blank line
O	Create a new blank line above the line in which the cursor is currently located and move the cursor to the beginning of the new blank line
r	Replace a single character at the cursor's current position
R	Replace multiple characters starting at the cursor's current position and ending when exiting insert mode
<Esc>	Exit insert or visual mode
> or <Tab>	Indent to the right the region selected
<	Indent to the left the region selected
v	Start highlighting characters
V	Start highlighting entire lines
yy	Yank/Copy the current line into memory
10yy	Yank/Copy 10 lines starting with the current line into memory
p	Put text yanked or deleted; if characters were yanked (`yw` `dw`, or `D`), put the characters after the cursor's current position. If lines were yanked, put the lines below the cursor's current line.
P	Same as p, but place characters before the cursor's current position or lines above the cursor's current line
u	Undo the last change
<Ctrl> R	Redo
/<pattern>	Search for the next pattern found, and place the cursor at the beginning of the pattern found
?<pattern>	Search for the previous pattern found, and place the cursor at the beginning of the pattern found
n	Repeat the last search
N	Repeat the last search, but reverse the search direction

!<cmd>	Execute <`cmd`> outside the Vim session

Line mode

Although command and insert modes are widely used, line mode is equally powerful but sometimes not fully understood or used. Line mode enters into a line editor, allowing you to process commands on single or multiple lines. Considering that vi was named after the ex editor, it's only fitting that line mode puts you into an ex editor.

To enter line mode, from command mode, simply type a colon (`:`). The cursor then moves to the lower-left corner of the window. When you resume typing, all text appears after the colon at the bottom of the window. When you click **Enter**, the line mode command is evaluated and executed. If you decide not to execute the line entered in line mode, click **Escape** to return to command mode.

When working with the line mode of the editor, keep two styles of commands in mind. First, when you type a command, vi or Vim executes the command as is. If the command executed pertains to modifying data, the current line will be the target. However, with the second method, you can supply line numbers to process the specified lines. To enter lines, after the colon, type the line number or range of lines to process, separating the start and end range with a comma (`,`).

For example, to process only line 23, the command would begin with **`:23`**. If you want to modify lines 2319 through 3819, you would type **`:2319,3819`**. To process a command from a starting position of line 45 to the end of the file, replace the last line argument with a dollar sign (**`$`**)—that is, **`:45,$`**.

The following commands are only the beginning of what you can execute in line mode:

- **`:w <file name>`:** Write the file to disk. If an argument is supplied, the editor attempts to write the data to <file name>.
  Note: If you supply an argument and <file name> exists, the editor will not overwrite the existing file's data.
- **`:w! <file name>`:** Write the file to disk and overwrite any data in the file.
- **`:<x>,<y> w <file name>`:** Write lines <x> through <y> to <file name>.
- **`:q`:** Attempt to exit the editor without saving.

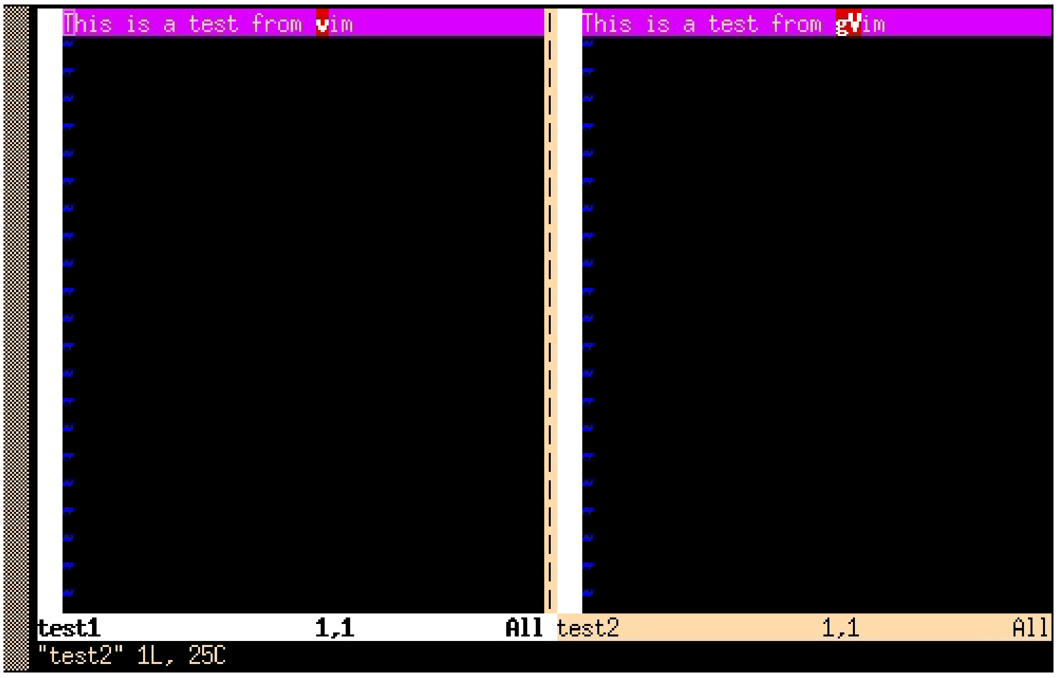
Note: If data has been modified, the editor will not exit until the file has been saved or you exit without saving.

- **`:q!`:** Exit the editor without attempting to write the file to disk.
- **`:n`:** If editing multiple files, begin editing the next file in the edit list.
- **`:e <file name>`:** Edit <file name>.
- **`:e#`:** If editing two files, switch between files.
- **`:s/<str1>/<str2>/`:** Replace the first occurrence of <str1> with <str2> on the current line.
- **`:1,$ s/<str1>/<str2>/g`:** Starting at line 1 and continuing throughout the file, replace <str1> with <str2> globally.
- **`:r <file name>`:** Read <file name> into the current editor session.
- **`:<x>,<y> d`:** Delete lines <x> through <y>.
- **`:<x>,<y> y`:** Yank lines <x> through <y>.

Differences between vi and Vim

Although vi and Vim do have the same look and feel, they definitely have differences. The following are just a few differences between these two powerful text editors:

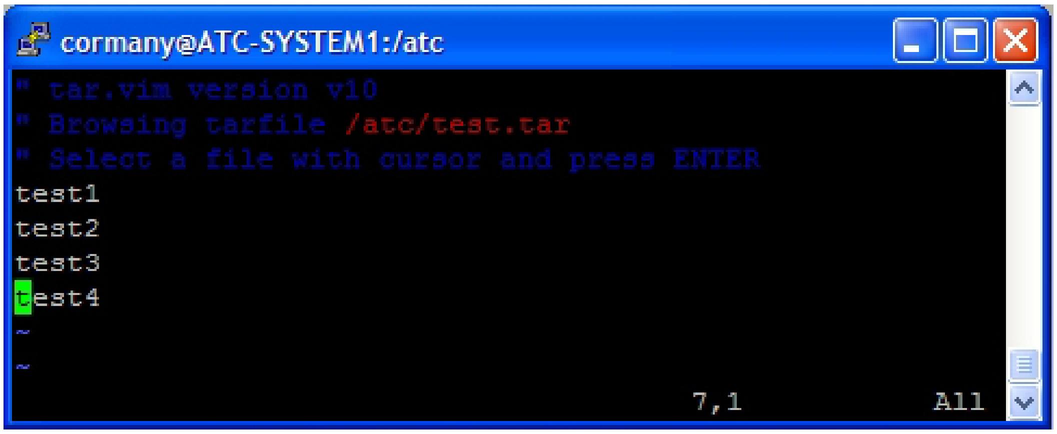
- **vimscript:** Using the internal scripting language vimscript, Vim allows complex scripts to add extended functionality to the editor. In addition to vimscript, Vim supports Perl, Python, Ruby, Tcl, and other languages.
- **vimdiff:** A useful command called `vimdiff` is bundled with the Vim package. Using `vimdiff`, you can display multiple files next to each other, similar to `sdiff`, as shown in Figure 6.
  Figure 6. An example of vimdiff



```
This is a test from vim          | This is a test from gVim
test1            1,1         All   test2           1,1        All
"test2" 1L, 25C
```

- **Editing a compressed file:** To conserve space on a system, administrators often compress log files or other large files. It never fails: A file is compressed, and then someone asks you, "Hey, can you take a look at this log from two months ago?" Rather than decompressing the file, and then editing it with vi, Vim can edit the compressed file. Vim can handle editing files compressed with bzip2, gzip, and zip.
- **Editing an archived file:** Vim also has the ability to edit files concatenated by `tar`. When editing a .tar file, Vim displays a handy screen allowing you to select which file in the archive you want to edit, as shown in Figure 7. When you finish editing the file, simply save and exit the file normally (`:wq`), and Vim returns to the display, allowing you to select another file to edit in the archive, or you can quit from the selection window (`:q`).

Figure 7. Selecting which file to edit in an achieved file using Vim

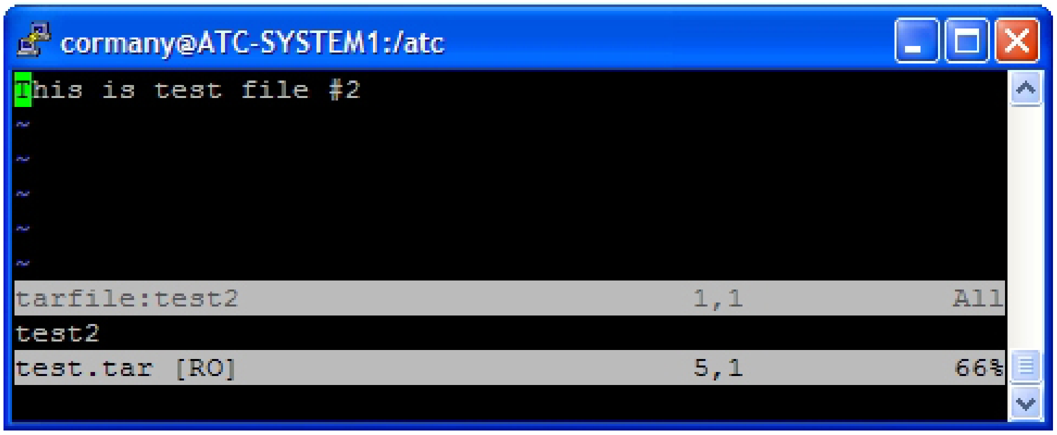


cormany@ATC-SYSTEM1:/atc

```
" tar.vim version v10
" Browsing tarfile /atc/test.tar
" Select a file with cursor and press ENTER
test1
test2
test3
test4
~
~
                                        7,1           All
```

In the example shown in Figure 8, four ASCII text files were archived using the `tar` command, and then Vim was used to select the second file in the archive to edit.

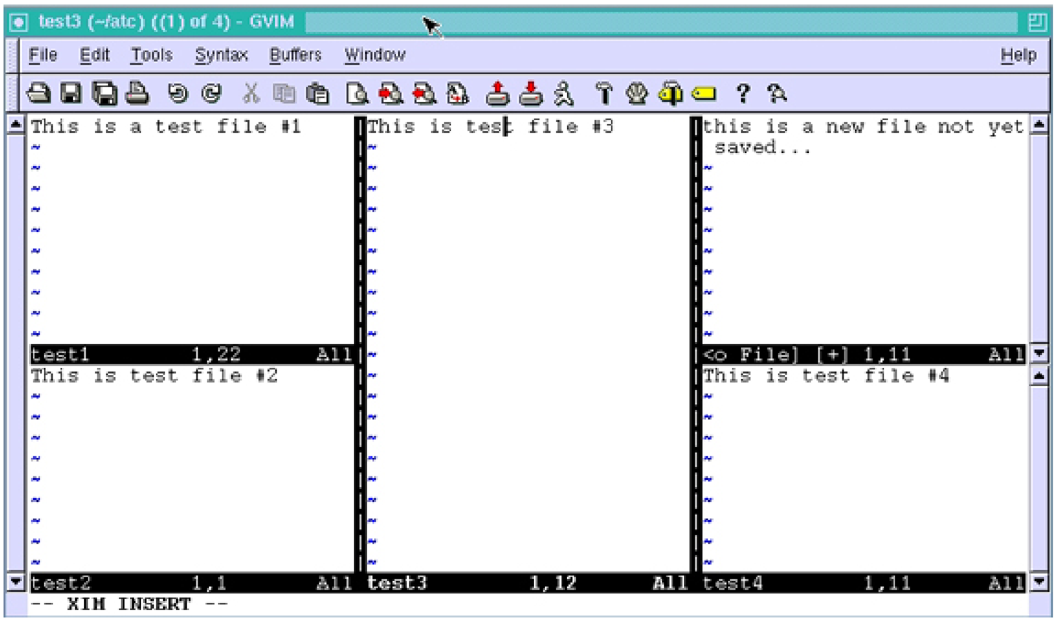
Figure 8. Editing a file within an archive through Vim



```
This is test file #2
~
~
~
~
~
tarfile:test2                                   1,1          All
test2
test.tar [RO]                                   5,1          66%
```

- **Split windows:** Rather than switching back and forth from window to window while editing multiple files, gVim allows you to open several windows from existing files as well as create new files on the fly, as Figure 9 shows.

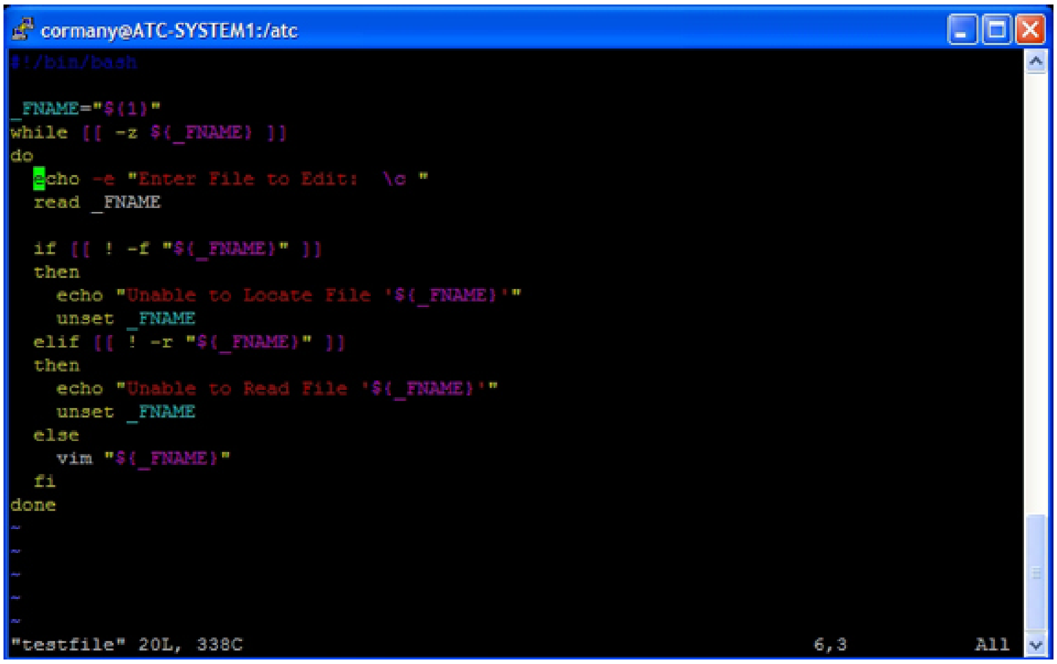
Figure 9. Split windows in Vim


test3 (~/atc) ((1) of 4) - GVIM
File Edit Tools Syntax Buffers Window Help

```
This is a test file #1    This is test file #3    this is a new file not yet
                                                   saved...
test1        1,22    All                          <o File] [+] 1,11     All
This is test file #2                              This is test file #4
test2        1,1     All  test3       1,12   All  test4       1,11      All
-- XIM INSERT --
```

- **Syntax highlights:** Debugging someone's shell script or other code that the editor didn't write originally can feel like an overwhelming task. Thankfully, Vim has helped alleviate some of the headaches that come with editing someone else's code. Using Vim, you can color-coad blocks of code, making debugging much easier in shell scripts as well as in other programming languages, as Figure 10 shows.

Figure 10. Syntax highlights



```
#!/bin/bash

_FNAME="${1}"
while [[ -z ${_FNAME} ]]
do
  echo -e "Enter File to Edit:  \c "
  read _FNAME

  if [[ ! -f "${_FNAME}" ]]
  then
    echo "Unable to Locate File '${_FNAME}'"
    unset _FNAME
  elif [[ ! -r "${_FNAME}" ]]
  then
    echo "Unable to Read File '${_FNAME}'"
    unset _FNAME
  else
    vim "${_FNAME}"
  fi
done
~
~
~
~
~
"testfile" 20L, 338C                                                                    6,3           All
```

- **Last cursor position:** When editing a file, it's sometimes necessary to exit the file and perform other tasks. But when you're ready to return to the file, you've forgotten where you left off! Not to worry: Vim remembers the last cursor position when exiting a file. This becomes extremely helpful when modifying files that are several thousands of lines long.
- **Multiple undo/redo operations:** In the past, vi only allowed you to undo your last change when editing a file. This was a great start, but it needed to be increased. Sometimes, when writing scripts or other code, what may seem like a good idea turns out not to be the best way to handle an issue, so you must be able to back out of the last 10 changes you've made in the file. Vim allows you to do just this.
- **Visual mode:** Vim allows for certain blocks of text to be selected using "visual" mode. Using this mode, you can select any amount of text within the file being edited, and then use a single command to affect the highlighted text. For example, if the middle of a paragraph of text must be removed, you simply type v to enter visual mode, move the cursor through the text to be modified, then type d to delete the highlighted text.

How do I get Vim?

Now that you've seen a few of the enhancements and differences between vi and Vim, you're probably saying, "I want Vim! Where do I get it?" Good news: Vim has been ported to several different operating systems.

Here are just a few of the platforms to which Vim has been ported:

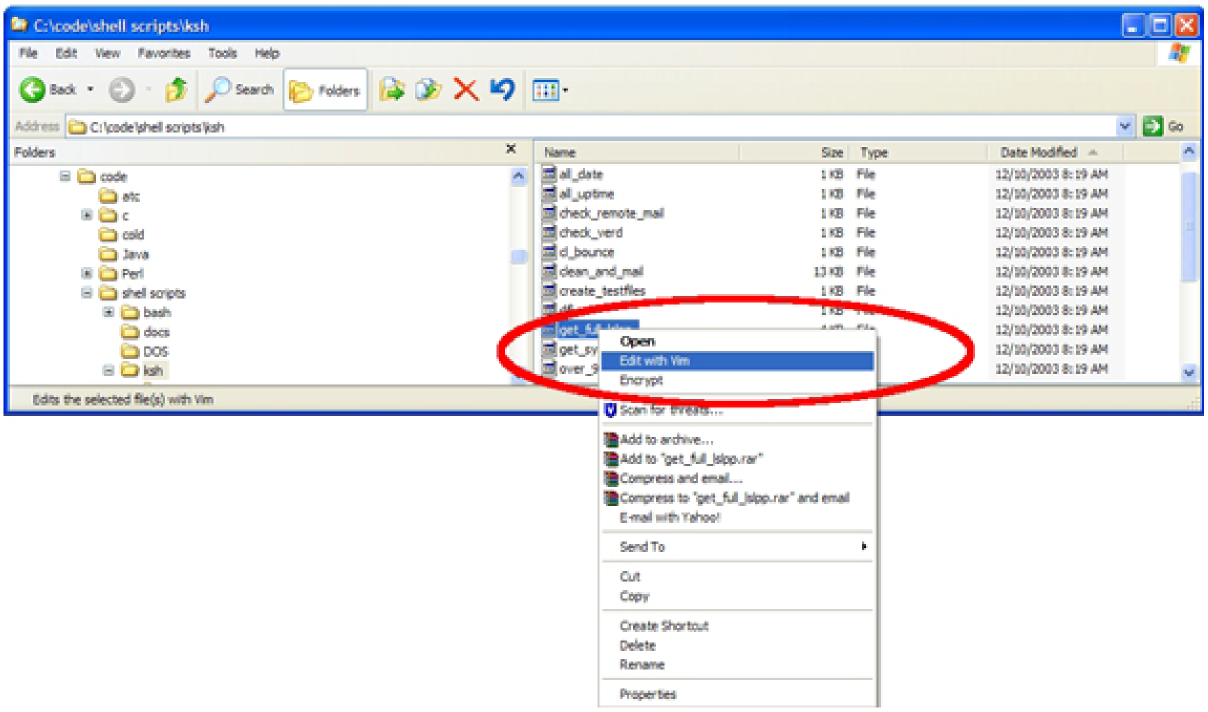
- Amiga
- AIX
- BSD
- Cygwin
- IBM OS/2®
- Linux
- Apple Mac OS and Mac OS X
- Microsoft® MS-DOS®
- Microsoft Windows® 95 though Windows Vista®
- Microsoft Windows CE
- OpenVMS

Vim on Windows

So, you saw that Vim is available for Windows in the previous section, and now you're saying, "I can use Vim on Windows? I want that!" You're in luck!

Simply download the latest version (currently version 7.1) of Vim, ported to Windows, from the Vim Web site. The easiest method is to download the self-installing executable file, execute it, and follow the steps. When installed, you can right-click a file, click **Edit with Vim** (as shown in Figure 11 below, and violà! you're now editing the file in Vim in Windows!

Figure 11. Editing a file with Vim in Windows



Name	Size	Type	Date Modified
all_date	1 KB	File	12/10/2003 8:19 AM
all_uptime	1 KB	File	12/10/2003 8:19 AM
check_remote_mail	1 KB	File	12/10/2003 8:19 AM
check_verd	1 KB	File	12/10/2003 8:19 AM
cl_bounce	1 KB	File	12/10/2003 8:19 AM
clean_and_mail	13 KB	File	12/10/2003 8:19 AM
create_testfiles	1 KB	File	12/10/2003 8:19 AM
dif	1 KB	File	12/10/2003 8:19 AM
get_full_lslpp		File	12/10/2003 8:19 AM
get_sy			12/10/2003 8:19 AM
over_			12/10/2003 8:19 AM

Now that you've installed Vim on your Windows computer, you can enjoy the genius behind Vim and forget about all the other text editors in Windows.

Conclusion

The vi editor started to pave the way for text editors in UNIX, and Vim has continued along this path. After reading this article, my hope is that you've learned some new things about the Vim editor, how to use its many features to help make your life easier, and appreciate how such a simple concept as an editor has proven a stable and robust application in the UNIX world and now most other operating systems. I trust that if you haven't used Vim much, after learning the ins and outs of the editor, you'll agree that no other editor can compare.

Resources

Learn

- Speaking UNIX: Check out other parts in this series.
- Wikipedia's AIX entry: Read Wikipedia's excellent entry on the AIX operating system for more information about its background and development.
- Wikipedia's vi editor entry: Read Wikipedia for more information about the vi editor.
- Wikipedia's VIM editor entry: Read Wikipedia's excellent entry on the Vim editor.
- The Vim editor: Learn more about the Vim editor.
- The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- New to AIX and UNIX? Visit the New to AIX and UNIX page to learn more.
- developerWorks technical events and webcasts: Stay current with developerWorks technical events and webcasts.
- AIX Wiki: Visit this collaborative environment for technical information related to AIX.
- Podcasts: Tune in and catch up with IBM technical experts.

Get products and technologies

- IBM trial software: Build your next development project with software for download directly from developerWorks.

Discuss

-
    - AIX Forum
    - AIX Forum for Developers
    - Cluster Systems Management
    - IBM Support Assistant Forum
    - Performance Tools Forum
    - Virtualization Forum
    - More AIX and UNIX forums

About the author

Adam T. Cormany
Adam Cormany is currently the manager of the National Data Center, but he has also been a UNIX systems engineer, a UNIX administrator, and operations manager for Scientific Games Corporation. Adam has worked extensively with AIX as well as in Solaris and Red Hat Linux administration for more than 10 years. He is an IBM eServer®-Certified Specialist in pSeries® AIX System Administration. In addition to administration, Adam has extensive knowledge of shell scripting in Bash, CSH, and KSH as well as programming in C, PHP, and Perl. You can reach Adam at acormany@yahoo.com.

Trademarks