Lecture 23: vi editor

Vi (pronounce: ``vee eye", not ``six", not ``vye") is an editor.
An editor is a program to edit files.

The VI editor is a screen-based editor used by many Unix users. The VI editor has powerful features to aid programmers, but many beginning users avoid using VI because the different features overwhelm them. Although other stories exist, the true one tells that Vi was originally written by Bill Joy in 1976. Bill took the sources of *ed* and *ex*, two horrendous programs for Unix that try to enable a human being to edit files, and created Vi.

Should I use Vi?

Which editor to use is mainly a matter of taste, style, and needs. Big chance that Vi is OK for---at least---the last one.
The long story is that, even though Vi is somewhat awkward to use at first, it enables fast, simple, and effective editing once you get the hang of it. A key concept in Vi is combining a certain *action* (delete, copy to buffer, capitalize, etc.) with a *movement* (go to line 25, go to end of document, go to next occurrence of ``foo," go to 2nd occurrence of character ``x" in this line, etc.). The action is performed on all lines or characters between the current cursor position and the destination cursor position. Vi is extremely powerful in moving around within (or between) files---Vim in particular is excellent. You can jump to a specific line, to the line where you were before jumping to the current line, to the line in the middle of the screen, to the line where you just changed ``foo" into ``bar," etc. You'll never have to mess with arrow keys to move around within a file. Finally, I observe that an effective Vi user simply edits files faster than Emacs people. Last but not least, you don't need a third hand (or nose) to type impossible key combinations. Don't get me wrong: Emacs is a great operating system---it lacks a good editor, though.

Vi has its dark sides, too. The biggest one is the need to step back before leaping forward when you are new to Vi. You cannot use Vi properly before knowing at least a handful of commands. This makes the threshold rather high.

Vi versions

Look at the filename to get some idea about the version...

Unix, MS-DOS, Windows, and OS/2

	Unix	MS-DOS	Windows 3.x	Windows 9x/2k/NT/XP	OS/2
VIM[1]	vim-6.1.tar.bz2	vim61d16.zip	*Use MS-DOS*	gvim61.zip	vim61os2.zip
Elvis[2]	elvis-2.1_4.tar.gz	elvis-2.1_4-msdos.tar.gz	*Not Available*	elvis-2.1_4-win32.tar.gz	elvis-2.1_4-os2.tar.gz
VILE	vile-9.2.tgz	vile-dos.zip	*Not Available*	vile-w32.zip	vile-os2.zip
Lemmy	*Not Available*	*Not Available*	*Not Available*	lemmy42.exe	*Not Available*
Nvi	nvi-1.79.tar.gz	*Not Available*	*Not Available*	*Not Available*	*Not Available*
Stevie	*Not Available*	stevie69x.zip[3]	*Not Available*	*Not Available*	stvi369g.zip
WinVi	*Not Available*	*Not Available*	winvi16.zip	winvi32.zip	*Not Available*
xvi	*Not Available*	xviexe.zip	*Not Available*	xvi.zip	*Not Available*
Pvic	*Not*	pvic_dos.zip	*Not*	*Not Available*	*Not Available*

	Available		*Available*		
Calvin	*Not Available*	calvin23.zip	*Not Available*	*Not Available*	*Not Available*

Macintosh, Atari, Amiga and OpenVMS

	Macintosh	Atari	Amiga	OpenVMS/Alpha	OpenVMS/VAX
VIM	vim5.6.full.fat.sit	vim-4.6.mint.bin.tgz	vim56bin.tgz	vim-56-alpha.zip	vim-56-vax.zip
VILE	*Not Available*	*Not Available*	*Not Available*	vile-9.2	vile-9.2
Stevie	stevie3.69+.sit.hqx	stevie.lzh[4]	Stevie.lha	*Not Available*	*Not Available*

Mastering the VI editor

EX Commands

Before You Begin

The VI editor uses the full screen, so it needs to know what kind of terminal you have. The prompt looks like this:

TERM = (vt100)

If you know your terminal is a vt100 (or an emulator that can do vt100), just hit return for the terminal type when you log in. If you have an hp terminal, type "hp" for the terminal type and hit return. If you are not sure what kind of terminal you have, ask a lab monitor, or have someone help you set the correct terminal type.

If you make an error when you log in and type the wrong terminal type, don't panic and log out. You can type the following commands to fix the settings:

First, tell your shell what type of terminal you have. (If you're not sure what your shell is, type this command to see what shell you have: echo $SHELL.) For the

examples given, the terminal type is "vt100". Substitute it with whatever terminal type you have. For C shell (/bin/csh), the command is this:

```
set term=vt100
```

For Bourne Shell (/bin/sh) or Korn Shell (/bin/ksh), the commands are the following:

```
export TERM
TERM=vt100
```

Next, reset your terminal with this command:

```
tset
```

Now that the terminal type is (hopefully) correctly set, you are ready to get started with VI.

Starting the VI Editor

The VI editor lets a user create new files or edit existing files. The command to start the VI editor is vi, followed by the filename. For example to edit a file called *temporary*, you would type vi temporary and then return. You can start VI without a filename, but when you want to save your work, you will have to tell VI which filename to save it into later. When you start VI for the first time, you will see a screen filled with tildes (A tilde looks like this: ~) on the left side of the screen. Any blank lines beyond the end of the file are shown this way. At the bottom of your screen, the filename should be shown, if you specified an existing file, and the size of the file will be shown as well, like this:
"filename" 21 lines, 385 characters
If the file you specified does not exist, then it will tell you that it is a new file, like this:
"newfile" [New file]
If you started VI without a filename, the bottom line of the screen will just be blank when VI starts. If the screen does not show you these expected results, your terminal type may be set wrong. Just type :q and return to get out of VI, and fix your terminal type. If you don't know how, ask a lab monitor.

Getting Out of VI

Now that you know how to get into VI, it would be a good idea to know how to get out of it. The VI editor has two modes and in order to get out of VI, you have to be in *command* mode. Hit the key labeled "**Escape**" or "**Esc**" (If your terminal does not have such a key, then try ^[, or control-[.) to get into *command* mode. If you were already in the command mode when you hit "**Escape**", don't worry. It might beep, but you will still be in the *command* mode.

The command to quit out of VI is :q. Once in *command* mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of VI without saving is :q!. This lets you exit VI without saving any of the changes.

Of course, normally in an editor, you would want to save the changes you have made. The command to save the contents of the editor is :w. You can combine the above command with the quit command, or :wq. You can specify a different file name to save to by specifying the name after the :w. For example, if you wanted to save the file you were working as another filename called *filename2*, you would type: w filename2 and return.

Another way to save your changes and exit out of VI is the ZZ command. When in *command* mode, type ZZ and it will do the equivalent of :wq. If any changes were made to the file, it will be saved. This is the easiest way to leave the editor, with only two keystrokes.

The Two Modes of VI

The first thing most users learn about the VI editor is that it has two modes: *command* and *insert*. The *command* mode allows the entry of commands to manipulate text. These commands are usually one or two characters long, and can be entered with few keystrokes. The *insert* mode puts anything typed on the keyboard into the current file.

VI starts out in *command* mode. There are several commands that put the VI editor into *insert* mode. The most commonly used commands to get into insert mode are a and i. These two commands are described below. Once you are in *insert* mode, you get out of it by hitting the **escape** key. If your terminal does not have an **escape** key, ^[should work (control-[). You can hit escape two times in a row and VI would definitely be in

command mode. Hitting **escape** while you are already in *command* mode doesn't take the editor out of *command* mode. It may beep to tell you that you are already in that mode.

How to Type Commands in Command Mode

The command mode commands are normally in this format: (Optional arguments are given in the brackets)

[*count*] command [*where*]

Most commands are one character long, including those which use control characters. The commands described in this section are those which are used most commonly the VI editor.

The *count* is entered as a number beginning with any character from 1 to 9. For example, the x command deletes a character under the cursor. If you type 23x while in *command* mode, it will delete 23 characters.

Some commands use an optional *where* parameter, where you can specify how many lines or how much of the document the command affects, the *where* parameter can also be any command that moves the cursor.

Some Simple VI Commands

Here is a simple set of commands to get a beginning VI user started. There are many other convenient commands, which will be discussed in later sections.

a

enter *insert* mode, the characters typed in will be inserted after the current cursor position. If you specify a count, all the text that had been inserted will be repeated that many times.

h

move the cursor to the left one character position.

i

enter *insert* mode, the characters typed in will be inserted before the current cursor position. If you specify a count, all the text that had been inserted will be repeated that many times.

j

move the cursor down one line.

k

move the cursor up one line.

l

move the cursor to the right one character position.

r

replace one character under the cursor. Specify *count* to replace a number of characters

u

undo the last change to the file. Typing u again will re-do the change.

x

delete character under the cursor. *Count* specifies how many characters to delete. The characters will be deleted after the cursor.

Text Buffers in VI

The VI editor has 36 buffers for storing pieces of text, and also a general purpose buffer. Any time a block of text is deleted or yanked from the file, it gets placed into the general purpose buffer. Most users of VI rarely use the other buffers, and can get along without the other buffers. The block of text is also stored in another buffer as well, if it is specified. The buffer is specified using the " command. After typing ", a letter or digit specifying the buffer must be entered. For example, the command: "mdd uses the buffer **m**, and the last two characters stand for delete current line. Similarly, text can be pasted in with the p or P command. "mp pastes the contents of buffer **m** after the current cursor position. For any of the commands used in the next two sections, these buffers can be specified for temporary storage of words or paragraphs.

Cutting and Yanking

The command commonly used command for cutting is d. This command deletes text from the file. The command is preceded by an optional *count* and followed by a movement specification. If you double the command by typing dd, it deletes the current line. Here are some combinations of these:

d^

deletes from current cursor position to the beginning of the line.

d$

deletes from current cursor position to the end of the line.

dw

deletes from current cursor position to the end of the word.

3dd

deletes three lines from current cursor position downwards.

There is also the y command which operates similarly to the d command which take text from the file without deleting the text.

Pasting

The commands to paste are p and P. The only differ in the position relative to the cursor where they paste. p pastes the specified or general buffer after the cursor position, while P pastes the specified or general buffer before the cursor position. Specifying *count* before the paste command pastes text the specified number of times.

Indenting Your Code and Checking

The VI editor has features to help programmers format their code neatly. There is a variable that to set up the indentation for each level of nesting in code.
The following commands indent your lines or remove the indentation, and can be specified with *count*:

<<

Shifts the current line to the left by one shift width.

>>

Shifts the current line to the right by one shift width.

The VI editor also has a helpful feature which checks your source code for any hanging parentheses or braces. The % command will look for the left parenthesis or brace corresponding to a particular right parenthesis or brace and vice versa. Place the cursor onto a parenthesis or brace and type **%** to move the cursor to the corresponding parenthesis or brace. This is useful to check for unclosed parentheses or braces. If a parenthesis or brace exists without a matching parenthesis or brace, VI will beep at you to indicate that no matching symbol was found.

Word and Character Searching

The VI editor has two kinds of searches: string and character. For a string search, the / and ? commands are used. When you start these commands, the command just typed will be shown on the bottom line, where you type the particular string to look for. These two commands differ only in the direction where the search takes place. The / command searches forwards (downwards) in the file, while the ? command searches backwards (upwards) in the file. The n and N commands repeat the previous search command in the same or opposite direction, respectively. Some characters have special meanings to VI, so they must be preceded by a backslash (\) to be included as part of the search expression. Special characters:

^

Beginning of the line. (At the beginning of a search expression.)

.

Matches a single character.

*

Matches zero or more of the previous character.

$

End of the line (At the end of the search expression.)

[

Starts a set of matching, or non-matching expressions... For example: /f[iae]t matches either of these: fit fat fet In this form, it matches anything except these:

/a[^bcd] will not match any of these, but anything with an a and another letter: ab ac ad

<

Put in an expression escaped with the backslash to find the ending or beginning of a word. For example: /\<the\> should find only word the, but not words like these: there and other.

>

See the '<' character description above.

The character search searches within one line to find a character entered after the command. The f and F commands search for a character on the current line only. f searches forwards and F searches backwards and the cursor moves to the position of the found character.

The t and T commands search for a character on the current line only, but for t, the cursor moves to the position before the character, and T searches the line backwards to the position after the character.

These two sets of commands can be repeated using the ; or , command, where ; repeats the last character search command in the same direction, while , repeats the command in the reverse direction.

Settings for VI (and EX)

You can customize the way VI behaves upon start up. There are several edit options which are available using the :set command, these are the VI and EX editor options available on Wiliki: (You can get this list by typing :set all and then **return** in command mode)

noautoindent magic noshowmatch
autoprint mesg noshowmode
noautowrite nomodelines noslowopen
nobeautify nonumber tabstop=8
directory=/tmp nonovice taglength=0

```
nodoubleescape      nooptimize                  tags=tags /usr/lib/tags
noedcompatible      paragraphs=IPLPPPQPP LIpplpipnpbp   term=xterm
noerrorbells        prompt                      noterse
noexrc              noreadonly                  timeout
flash               redraw                      timeoutlen=500
hardtabs=8          remap                       ttytype=xterm
noignorecase        report=5                    warn
keyboardedit        scroll=11                   window=23
keyboardedit!       sections=NHSHH HUuhsh+c     wrapscan
nolisp              shell=/bin/csh              wrapmargin=0
nolist              shiftwidth=8                nowriteany
```

Some of these options have values set with the equals sign '=' in it, while others are either set or not set. (These on or off type of options are called **Boolean**, and have "no" in front of them to indicate that they are not set.) The options shown here are the options that are set without any customization. Descriptions of some of these are given below, with an abbreviation. For example, the command set autoindent, you can type :set autoindent or :set ai. To unset it, you can type :set noautoindent or :set noai.

autoindent (ai)

This option sets the editor so that lines following an indented line will have the same indentation as the previous line. If you want to back over this indentation, you can type ^D at the very first character position. This ^D works in the *insert* mode, and not in *command* mode. Also, the width of the indentations can be set with **shiftwidth**, explained below.

exrc

The *.exrc* file in the current directory is read during startup. This has to be set either in the environment variable **EXINIT** or in the *.exrc* file in your home directory.

mesg

Turn off messages if this option is unset using :set nomesg, so that nobody can bother you while using the editor.

number (nu)

Displays lines with line numbers on the left side.

shiftwidth (sw)

This option takes a value, and determines the width of a software tabstop. (The software tabstop is used for the << and >> commands.) For example, you would set a shift width of 4 with this command: :set sw=4.

showmode (smd)

This option is used to show the actual mode of the editor that you are in. If you are in *insert* mode, the bottom line of the screen will say **INPUT MODE**.

warn

This option warns you if you have modified the file, but haven't saved it yet.

window (wi)

This option sets up the number of lines on the window that VI uses. For example, to set the VI editor to use only 12 lines of your screen (because your modem is slow) you would use this: :set wi=12.

wrapscan (ws)

This option affects the behavior of the word search. If wrapscan is set, if the word is not found at the bottom of the file, it will try to search for it at the beginning.

wrapmargin (wm)

If this option has a value greater than zero, the editor will automatically "word wrap". That is, if you get to within that many spaces of the left margin, the word will wrap to the next line, without having to type return. For example, to set the wrap margin to two characters, you would type this: :set wm=2.

Abbreviations and Mapping Keys to Other Keys

One EX editor command that is useful in the VI editor is the **abbreviate** command. This lets you set up abbreviations for specific strings. The command looks like this: :ab string

thing to substitute for. For example, if you had to type the name, "**Humuhumunukunukuapua`a**" but you didn't want to type the whole name, you could use an abbreviation for it. For this example, the command is entered like this:

:ab 9u Humuhumunukunukuapua`a

Now, whenever you type 9u as a separate word, VI will type in the entire word(s) specified. If you typed in 9university, it will not substitute the word.

To remove a previously defined abbreviation, the command is unabbreviate. To remove the previous example, the command would be ":una 9u" To get your listing of abbreviations, simply just type :ab without any definitions.

Another EX editor command that is useful for customization is the mapping command. There are two kinds of mapping commands. One for command mode, and the other for insert mode. These two commands are :map and :map! respectively. The mapping works similarly to the abbreviation, and you give it a key sequence and give it another key sequence to substitute it with. (The substituted key sequences are usually VI commands.)

The EXINIT Environment Variable and the *.exrc* file

There are two ways to customize the VI editor. If you create a file called *.exrc* in your home directory, all the commands in there will be read when VI starts up. The other method is to set an environment variable called **EXINIT**. The options will be set in your shell's setup file. If you use /bin/csh (C-Shell), the command is as follows, and is put in the *.cshrc* file:

```
setenv EXINIT '...'
```

If you use /bin/sh or /bin/ksh, the command is as follows, and is put into the *.profile* file:

```
export EXINIT
EXINIT='...'
```

Don't put in ... as the example says. In this space put the commands that you want to set up. For example, if you want to have auto indent, line numbering, and the wrap margin of three characters, then the setenv command (for C shell) looks like this:

```
setenv EXINIT 'set ai nu wm=3'
```

If you want to put more than one command in the setenv EXINIT thing, separate the commands with a vertical bar (|). For example, to map the 'g' command to the 'G'

character in command mode, the command is :map g G, and combined with the above command, you get this:
setenv EXINIT 'set ai nu wm=3|map g G'
If you want to create the file called *.exrc*, you can put exactly the same things in the file as shown in the quotes after the **EXINIT**.

Recovering Your Work When Something Goes Wrong with Your Terminal

The VI editor edits a temporary copy of your file, and after the editing is complete, or when you tell it to save, it puts the contents of the temporary copy into the original file. If something goes wrong while you are editing your file, the VI editor will attempt to save whatever work you had in progress, and store it for later recovery. (Note: If VI dies while you were working on any file, it sends you an email message on how to recover it. The **-r** option stands for recovery. If you were editing the file *vitalinfo*, and you accidentally got logged out, then the **-r** option of the 'vi' editor should help. The command would look somewhat like this: vi -r vitalinfo After using the **-r** option once, though, you MUST save what you have recovered to the actual file... The **-r** option only works once per failed VI session.

Warning About Using VI on the Workstations

There are two things to be aware of when using the workstations: Editing the same file many times at once, and changing the size of the screen.
Because VI edits a copy of your original file and saves the contents of that copy into the original file, if you are logged on more than once and are editing the same file more than once using VI, if you save on one window and then you save on the other window, the changes made to the file on the first save would be overwritten. Make sure that you only run one copy of VI per file.
If you use a terminal program from a workstation, you can change the size of the screen by dragging the sides of the window. If the size is not working properly, the command to type is this:

```
eval `resize`
```

If that doesn't work the command would be this:

```
eval `/usr/bin/X11/resize`
```

If the size is wrong, the editor will not operate correctly. If you have any problems with the screen size, ask the monitors in the computer lab for help setting the sizes correctly.

Summary of VI commands

This list is a summary of VI commands, categorized by function.

"

Specify a buffer to be used any of the commands using buffers. Follow the " with a letter or a number, which corresponds to a buffer.

D

Delete to the end of the line from the current cursor position.

P

Paste the specified buffer before the current cursor position or line. If no buffer is specified (with the " command.) then 'P' uses the general buffer.

X

Delete the character before the cursor.

Y

Yank the current line into the specified buffer. If no buffer is specified, then the general buffer is used.

d

Delete until *where*. "dd" deletes the current line. A count deletes that many lines. Whatever is deleted is placed into the buffer specified with the " command. If no buffer is specified, then the general buffer is used.

p

Paste the specified buffer after the current cursor position or line. If no buffer is specified (with the " command.) then 'p' uses the general buffer.

x

Delete character under the cursor. A count tells how many characters to delete. The characters will be deleted after the cursor.

y

Yank until, putting the result into a buffer. "yy" yanks the current line. a count yanks that many lines. The buffer can be specified with the " command. If no buffer is specified, then the general buffer is used.

Inser**ting New Text**

A

Append at the end of the current line.

I

Insert from the beginning of a line.

O

(letter oh) Enter *insert* mode in a new line above the current cursor position.

a

Enter *insert* mode, the characters typed in will be inserted after the current cursor position. A count inserts all the text that had been inserted that many times.

i

Enter *insert* mode, the characters typed in will be inserted before the current cursor position. A count inserts all the text that had been inserted that many times.

o

Enter *insert* mode in a new line below the current cursor position.

Moving the Cursor Within the File

^B

Scroll backwards one page. A count scrolls that many pages.

^D

Scroll forwards half a window. A count scrolls that many lines.

^F

Scroll forwards one page. A count scrolls that many pages.

^H

Move the cursor one space to the left. A count moves that many spaces.

^J

Move the cursor down one line in the same column. A count moves that many lines down.

^M

Move to the first character on the next line.

^N

Move the cursor down one line in the same column. A count moves that many lines down.

^P

Move the cursor up one line in the same column. A count moves that many lines up.

^U

Scroll backwards half a window. A count scrolls that many lines.

$

Move the cursor to the end of the current line. A count moves to the end of the following lines.

%

Move the cursor to the matching parenthesis or brace.

^

Move the cursor to the first non-whitespace character.

(

Move the cursor to the beginning of a sentence.

)

Move the cursor to the beginning of the next sentence.

{

Move the cursor to the preceding paragraph.

}

Move the cursor to the next paragraph.

|

Move the cursor to the column specified by the count.

+

Move the cursor to the first non-whitespace character in the next line.

-

Move the cursor to the first non-whitespace character in the previous line.

_

Move the cursor to the first non-whitespace character in the current line.

0

(Zero) Move the cursor to the first column of the current line.

B

Move the cursor back one word, skipping over punctuation.

E

Move forward to the end of a word, skipping over punctuation.

G

Go to the line number specified as the count. If no count is given, then go to the end of the file.

H

Move the cursor to the first non-whitespace character on the top of the screen.

L

Move the cursor to the first non-whitespace character on the bottom of the screen.

M

Move the cursor to the first non-whitespace character on the middle of the screen.

W

Move forward to the beginning of a word, skipping over punctuation.

b

Move the cursor back one word. If the cursor is in the middle of a word, move the cursor to the first character of that word.

e

Move the cursor forward one word. If the cursor is in the middle of a word, move the cursor to the last character of that word.

h

Move the cursor to the left one character position.

j

Move the cursor down one line.

k

Move the cursor up one line.

l

Move the cursor to the right one character position.

w

Move the cursor forward one word. If the cursor is in the middle of a word, move the cursor to the first character of the next word.