C. How to...

This is a collection of "How to" topics, each with short discussion. The following catagories of topics are available:

* Initialization - Change the default behavior of elvis
* Word Completion - Alter word completion via the **Tab** key
* Whitespace - The relationship between tabs and spaces
* Buffers - Cut buffers and edit buffers
* X-Windows - The "x11" user interface
* Windows95/98/NT - The "windows" user interface
* Miscellany - Things that didn't fit anywhere else
* Contacts - How to contact the authors

If you're using elvis to view this file, you can search for a topic simply by using the / command. To limit the search to topic lines only, begin the regular expression with "^<dt>.*". For example you could search for "spaces" in a topic line via the following:

```
/^<dt>.*spaces
```

Sometimes there are multiple topics that use the same word or phrase. If the first one that it finds isn't the one you want, then you can use the n command to find the next one.

In addition, the lib/elvis.ali script distributed with elvis contains a "howto" alias which loads this file and searches for a given topic. The aliases from that file are loaded automatically, so you should be able to display any topic in a a separate window via a command such as this:

```
:howto spaces
```

C.1 Initialization

Initialization for each file

The lib/elvis.arf file is executed after a file has been read; you can put file initialization commands there.

Make file initialization be language-sensitive

If the filename extension indicates the file's language, and the language is described in the lib/elvis.syn file, then you can use the knownsyntax() function to check the language, like this:

```
:if knownsyntax(filename) == "perl"
:then set ccprg="perl -c ($1?$1:$2) 2>&1 | perlerr"
```

(The perlerr perl script converts perl's error messages into a form that elvis can parse. It is given in the Tips chapter.)

You could also use the dirext() function instead of knownsyntax(); it doesn't depend on the lib/elvis.syn file. As a last resort, you might consider using the x flag of the :s command.

Change the default settings

Before the first buffer is loaded or the first window is created, and buffer/window options you set will be used as the defaults for the buffers/windows that are created later. In other words, although setting a buffer-specific option like equalprg or a window-specific option like ruler interactively will only affect that buffer or window, setting the same option in the lib/elvis.ini or ~/.exrc file changes the default for all buffers/windows.

However, the lib/elvis.brf and lib/elvis.arf files typically change some of those options, so the defaults might not be used very long before they're changed.

Have a separate ~/.exrc and ~/.elvisrc file under Unix

This is easy. The lib/elvis.ini file controls this, and the default version does it in a very convenient way: It looks first for .elvisrc and runs it if it exists; else it looks for .exrc and runs that.

If you want elvis to run both of them, then add the following line to your .elvisrc file:

```
:source ~/.exrc
```

C.2 Word Completion

Disable word completion on the ex command line

You need to set the inputtab option for the (Elvis ex history) buffer to "tab". You can do that via the following command:

```
:(Elvis ex history)set inputtab=tab
```

Disable identifier completion on the regular expression prompt line

This is similar to word completion on the ex command line. To disable it, you need to set the inputtab option for the (Elvis regexp history) buffer, like this:

```
:(Elvis regexp history)set inputtab=tab
```

Enable identifier completion in normal edit buffers

Set the buffer's inputtab option to "identifier". To make this the default, set it in the .exrc file (or elvis.rc for non-Unix systems). The command is:

```
:set inputtab=identifier
```

After that, each time you hit the <**Tab**> key elvis will search through the tags file for any matching tags, and add as many characters as possible. If it completes the entire tag name, it does *not* append a space or other character, which is a little different from other types of name completion. Also, in the syntax display mode it will not attempt completion if the partial word happens to be a complete keyword or otherword.

C.3 Whitespace

Use spaces instead of tabs

In the traditional vi, the only way to make elvis use spaces instead of tabs was to set the tabstop option to the largest possible value, and then :map ^I to ^T. This made existing tabbed files look bad, and it didn't work well for tabs embedded in a line (instead of just in the line's indentation whitespace). Elvis has a better way.

In elvis, you can set the inputtab option to "spaces" to make the **<Tab>** key insert the appropriate number of spaces into a line. This works even if the cursor isn't in the line's indentation whitespace.

In addition, the autotab option controls the shifting commands (the :< and :> commands in ex, and the ≤ and ≥ operators in visual mode). To make those commands use only spaces, *autotab* should be off.

```
:set inputtab=spaces noautotab
```

To convert existing files to use only spaces, you should use an external program such as "col -bx" under Unix.

Change the tabstops

Many people don't like the fact that the **<Tab>** key indents text by 8 columns. That's so wide that it quickly pushes the writer's source code off the right edge of the screen.

However, you almost certainly do *not* want to change the tabstop option because most other software, and most printers and terminals, also assume that tabs are 8 characters wide. If you edit files with tabstop set to 4 or 5, then your files will look very strange when viewed with anything other than elvis, or by anyone other than you. So leave tabstop=8.

Instead, set the shiftwidth option to the desired indentation amount, and either get in the habit of typing ^T to increase indentation, or :map! the **<Tab>** key to ^T in input mode.

```
:set shiftwidth=5
:map! ^V^I ^V^T
```

Note that when you're typing in the above :map command, you'll need to type an extra **^V** before each **^V** or **^T**.

Also, this map has the unfortunate side-effect of making the **<Tab>** increase indentation even if the cursor is somewhere later in the line (unless you type ^V before it). This is one good reason to skip the map, and get in the habit of using **^T** to increase indentation. The autotab option helps here, too.

C.4 Buffers

Switch to a different buffer in the same window

This is easier that you might think. On an ex command line, if you give an address but no command then elvis will move the cursor to there. So to switch buffers all you need to do is give an address that's in a different buffer. In elvis, you do this by giving the buffer's name (or number) in parentheses (and the closing parenthesis is optional). For example, to switch to buffer #1 all you need to do is...

```
:(1
```

Or you can switch to "main.c" like this:

```
:(main.c
```

Of course, the buffer must exist before you can switch to it. Another thing to keep in mind is, switching buffers doesn't necessarily force you to save the old buffer first. Any changes you made to the old buffer are *not* lost -- you can switch back to the original buffer again if you wish.

Display an edit buffer (or cut buffer) in a separate window

This is similar to switching edit buffers (the previous topic). The main difference is that instead of giving no command, you should give the :split command. In this context, the closing parenthesis is *required*.

```
:(1)split
```

Or, create a window showing the "main.c" buffer:

```
:(main.c)split
```

Edit a cut buffer

Editing a cut buffer can be handy when you're trying to fix a defective macro. This is possible in elvis, because elvis uses an ordinary edit buffer to store the contents of a cut buffer. The names of the cut buffers are of the form (Elvis cut buffer *X*), where *X* is the name of the cut buffer (a single letter or digit). Consequently, you could create a window showing cut buffer "a like this:

```
:(Elvis cut buffer a)split
```

Of course, the "a cut buffer must exist for this to work.

Since the name is so long, elvis supports a special short-hand notation for cut buffer names. In parentheses, if the first character is " and the remainder of the buffer name is a single letter, then elvis uses the buffer which contains that cut buffer's contents. The following command also creates a window showing the "a cut buffer:

```
:("a)sp
```

Elvis doesn't store "undo" versions for cut buffers, and you can't yank a buffer into itself. Other than that, editing should be pretty normal. The type of data in the buffer (characters, lines, or rectangle) is stored in an option named putstyle.

Free an edit buffer

Elvis has no command for discarding old edit buffers. Under some circumstances it will free them automatically, if they aren't being used. It rarely matters, though.

C.5 X-Windows

Run elvis in an xterm instead of creating a new window

You can force elvis to use the termcap interface by adding a -Gtermcap flag. If you do this often, you may wish to create a shell script, alias, or shell function which runs elvis with -Gtermcap. Here's an example of shell script:

```
#!/bin/sh
exec elvis -Gtermcap "$@"
```

If you *never* want to use the "x11" user interface, then you should probably reconfigure elvis to leave it it out. This will make elvis considerably smaller. To do this, go into the directory where elvis' source code resides and execute the following shell commands:

```
make clean
configure --with-x=no
make
```

Make the text cursor more visible under X11

For a notebook computer, the normal blinking cursor may be hard to see. You can make it stop blinking by adding the following command to your .exrc file:

```
:set blinktime=0
```

Indicate when elvis owns the current X11 selection

The cursor can be configured to have a special color whenever elvis owns the current selection. To do this, use the :color command to set both the foreground and background color of the cursor. The "background" color will be used when elvis owns the selection, and the "foreground" color will be used otherwise. Here's an example which turns the cursor green when elvis owns the selection:

```
:color cursor red on green
```

Change the default font size

In your .exrc file, you can set the normalfont, boldfont, and italicfont options to anything you want. These settings will override the defaults. If you set only the normalfont and leave the others unset, then elvis will derive the others from the normal font.

```
:set normalfont=7x14
```

If you just want to use a smaller size of the Courier font, you can use the :courier alias. It takes a single parameter: the point size of the font to use. The default font is 18-point Courier, and most systems also have 14-point Courier fonts which works well.

```
:courier 14
```

Set a font for the toolbar buttons.

By default, the X11 toolbar uses a font named "variable" which is usually an alias to a small, readable, variable-pitch font. If your X server has no font named "variable" then you must configure elvis to use a different font. You may also simply prefer a different font.

The simplest way to change the font is to pass elvis a -fc *fontname* parameter. I suggest you use that to experiment with the available fonts, to find one you like. (You can use the standard xlsfonts program to list the available fonts.)

To make the change permanent, you can either set the controlfont option in your ~/.exrc file, or you can set the elvis.control.font resource in your ~/.Xdefaults file.

Run a program from within elvis, in parallel, under X11

The tricky part here is that elvis tries to read the program's stdout and stderr, so the output can be displayed in elvis' window. To do that, Elvis would have to wait until after all text has been read from stdout/stderr... but you don't want elvis to wait! So to run in parallel, you must redirect the program's stdout/stderr to /dev/null, like this:

```
:!xeyes >/dev/null 2>&1 &
```

If you want to write data out to the program (:w !program) then it becomes even more complex. This is because pipes can only contain a finite amount of data, so when elvis is redirecting stdin as well as stdout/stderr, it uses a temporary file for stdin. Elvis deletes that file as soon as the program returns -- which, for a program run in parallel, happens immediately even though the program hasn't had a chance to read the data from that file yet. The solution is to write the data into a temporary file sequentially, and then start a parallel command line which runs the program and then deletes the temporary file, like this:

```
:w !cat >$$; (xv $$; rm $$) >/dev/null 2>&1 &
```

Yes, that's nasty. I plan to clean that up some day, by making elvis smart enough to avoid reading stdout/stderr when the command line ends with a '&' character.

Run an interactive program under X11

Sadly, this can't be done with elvis' "x11" user interface because elvis' windows don't have a built-in terminal emulator. This is expected to be added for elvis 2.2, but that doesn't help you *now*.

However, by using the "termcap" interface inside an xterm, you should be able to run interactive programs such as "crypt" or "pgpv" exactly as you can under vi. In an xterm (or any other terminal emulator), just run "elvis -Gtermcap" instead of plain "elvis".

C.6 Windows95/98/NT

Set the initial working directory

Create a shortcut to WinElvis, and then edit the shortcut's properties. The "Start in" property gives the program's initial working directory.

Change the working directory each time a file is loaded

First, let me say that I don't recommend this because the real vi doesn't behave like this, so it is likely to confuse some people.

But it you really want to do this, then you should add the following line to the end of your lib/elvis.arf file:

```
try cd (dirdir(filename))
```

Select whole lines in WinElvis

To select whole lines in WinElvis, move the mouse pointer to the window's left margin. The mouse pointer should change shape when you're in the margin. Click the left mouse button there to begin marking lines; hold the button as you move the mouse to the other end of the range of lines and then release it.

Change the icon for WinElvis

WinElvis contains two icons -- a low-color version of the Elvis Presley postage stamp, and a tiny photo of elvis. The stamp is closer to the Unix icon for elvis, so that's the default. If you prefer the photo, you can use it by creating a link to WinElvis, viewing the link's properties, and clicking the "Change Icon" button there.

C.7 Miscellany

Test for certain text within a file

This is sometimes handy in scripts and aliases. For example, the elvis.arf script uses this to detect mode lines and the "hash pling" header on other types of scripts, such as shell scripts which start with "#!/bin/sh".

One nice trick is to use the x flag of elvis' :s command. It not only detects text, but can incorporate that text into the commands. For example, to compute the total of all numbers in all lines, you could...

```
:set t=0
:%s/\<[[:digit:]]\+\>/let t = t + &/gx
:set t?
```

Note that this series of commands does not affect the edit buffer. The x flag prevents the substitution from taking place; the replacement text is executed instead.

You can also use the :try command to run a search command, and then use :then and :else to act differently depending on whether the search succeeded or not.

```
:try /Yow!
:then echo Zippy was here
:else echo Where in the world is Zippy the Pinhead?
```

You can also use the current("word") and line() functions to fetch the word at the cursor location, or a whole line, respectively.

```
:let w=current("word")
:let l=line(1)
```

Find the short name (or group name) of an option

Finding the long name of an option is easy, thanks to name completion. To find the short name of an option, or the name of its group, run the :set! command (with a !) and the long option name, followed by a ? character. (For non-Boolean options, the ? is optional.)

```
:set! wrapmargin?
```

This will produce output like "win.wm=0", indicating that the short name is "wm", the group name is "win" (so each window has its own margin), and the value is 0.

Recover files after a crash

This is described in the Sessions chapter of the manual. Briefly, run "elvis -r" to start a new elvis process on the old session file, and then use the :buffer command to list the buffers. You can then use other commands to save those buffers; for example, to save a buffer named "main.c" into a file named "main.c.recovered", you would give this command:

```
:(main.c)w main.c.recovered
```

Recognize error messages with an unusual format

Elvis' :make and :cc commands assume that all error messages have a format that resembles that of **gcc**: A file name and line number appear at the beginning of the line (possibly with some punctuation or the word "line" mixed in), an optional "error" or "warning" code, and then the description of the error to finish the line. Elvis is very good at parsing messages that use formats which resemble this, but there is no explicit way to make elvis parse any other format.

However, it is usually possible to construct a little "filter" program to convert other error message formats into one that elvis can recognize. The Tips chapter has an example of how to make elvis handle PERL's error messages.

Input non-ASCII characters

Elvis supports several ways to enter non-ASCII characters:

* If your keyboard driver supports entry of non-ASCII characters, then that method should work in elvis. However, "dead keys" have been reported to cause trouble.
* Digraphs combine two ASCII characters to form a single non-ASCII character. See the :digraph command, digraph option, and the discussion in the Input Mode chapter. Briefly, the secret is to type **<Ctrl-K>** and then the two ASCII characters that you want to combine, such as **<Ctrl-K><n><~>** for "ñ".
* You can type **<Ctrl-X>** followed by two hex characters to enter any byte into the edit buffer... provided you know the hex code for the character you want, of course.

If you're having trouble displaying non-ASCII characters, then you may want to look into the nonascii option. Also, on Unix systems you should verify that your terminal is configured correctly (8 bits, not 7 -- and the Latin-1 character set is installed).

Change the default address range for a command

This is sometimes desirable in an alias. For example the :w command writes all lines by default, but :s alters only one line by default. You may want to write a macro that uses both of them with the same default, or just one of them with a different default.

The most straightforward way to do this is to use the **!{***default***}%** notation. Specifically, **!{.}%** will make any command in an alias default to using just the current line, and **!{%}%** will make any command default to using all lines.

Here's a simple word-counting alias which uses this technique to count the words in all lines by default...

```
alias wcw {
 local w=0
 !{%}% s/\w\+/let w=w+1/gx
 calc w
}
```

Execute a particular command separately for each line in a range

The :g command does this, but only for the lines which contain a given regular expression. You can trick it into executing the command for every line by making it look for something that every line has. Since every line has a beginning, you can use **:g/^/***command* to execute the command for every line.

Note that the :g command can be used with an address range. This is often convenient.

Here's an alias which uses this technique to search for the longest line (assuming all characters are of equal width -- no tabs or control characters)...

```
alias widest {
 local w=0 l
 !%g/^/ {
  if strlen(line()) > w
  then let w = strlen(line())
  then let l = current("line")
 }
 calc "Line "l" is the longest, at "w" characters."
}
```

Convert plain text to HTML source

Other than the obvious (a long series of manual edit commands), there are two ways that you can convert text to HTML in elvis.

The most straightforward method is to build an alias which performs a series of substitutions. The problem with this method is that it must be rewritten for each type of input text. Here's an example of a fairly simple alias that converts plain text to HTML. (There is a more powerful version of this alias in the standard distribution's elvis.ali file.)

```
alias makehtml {
 "Convert plain text to HTML
 local report=0
 "
 "Protect characters which are special to
 try !(%)%s/&/\&amp;/g
 try !(%)%s/</\&lt;/g
 try !(%)%s/>/\&gt;/g
 "
 "Convert blank lines to <p> tags
 try !(%)s/^$/<p>/
 "
```

```
"If converting the whole file, then add <html>...</html>
if "!%" == ""
then {
 $a </body></html>
 1i <html><body>
}
}
```

The other way is simpler and more versatile, but it requires the use of an external file. It uses the :lpr command with lptype=html. Since elvis' print mechanism supports all of elvis' display modes, you can use this technique to convert any type of text (or even a hex dump of a binary file) into HTML.

```
:set lptype=html lplines=0 nolpheader
:lp foo.html
```

Add new text around highlighted text, via a macro.

The main trick is to use the visual c command to change the highlighted text, and then use **^P** as part of the replacement text. While typing the replacement text, **^P** causes a copy of the original text to be inserted. So, for example, the following macro could be used to and HTML **** and **** tags around the highlighted text.

```
:map B c<strong>^P</strong>^[
```

(Note: When typing that :map command into elvis, you'll need to type <Ctrl-V><Ctrl-P> to get the **^P**, and <Ctrl-V><Esc> to get the **^[** character.)

C.8 Contacts

Request technical support

The best way to get technical support is by posting a question on the comp.editors newsgroup. Be sure to mention "elvis" in the subject line, and try to include a succinct description of the problem there, too. In the body of the message, be sure to mention the version of elvis you're using (as reported by "elvis --version"), and your operating system.

Inform the authors about a bug

Bugs should be reported via e-mail. You may not always receive prompt confirmation, but your bug reports are appreciated, and acted-upon. For OS-specific bugs, you should contact the person who ported elvis to your operating system; the author's names and e-mail addresses are listed in the Operating Systems chapter.

For general bug reports, you should contact the primary author, Steve Kirkendall, at kirkenda@cs.pdx.edu.

Either way, be sure to mention the version of elvis you're using (as reported by "elvis --version"), and your operating system.

Suggest a new feature

Contact the authors as though you were reporting a bug. See above.

Get the latest version of elvis

See the "Information via the Web" section of the Tips chapter. It has links to various elvis-related sites.