16. TIPS

This section of the manual explains some of the more complex, obscure, and useful features of elvis (or larger subjects). The following subjects are discussed:

16.1 Information via the Web

Here are some URLs (World Wide Web links to other documents) which are relevant to vi. Each of these, in turn, has links to other sites. Directly or indirectly, these links will lead you to a huge amount of information about vi.

I've tried to limit this list to advertized sites; there are others that I run across from time to time, but their URLs tend to vary over time, so it isn't a good idea to place them in a static document such as this one. It makes for a short list, though.

ftp://ftp.cs.pdx.edu/pub/elvis/README.html
: The "home" site of elvis. This is where you can find the latest official release.

ftp://ftp.cs.pdx.edu/pub/elvis/unreleased/README.html
: Prerelease versions of elvis can often be found here. Prerelease versions are identified by a letter appended to the version number they they're expected to be released as. For example, 2.1a is a prerelease version of what will eventually be released as version 2.1. Also, I append "-alpha" in the early stages when new features are still being added, or "-beta" if no new features are expected before the release. Typically, new versions are uploaded once a month or so; watch for announcements on the comp.editors newsgroup.

http://www.fh-wedel.de/elvis/index.html
: This where Herbert (a.k.a. Martin Dietze, who ported elvis to OS/2) is putting an Elvis home page. It is expected to have links to all relevent web pages, archives, and people's email addresses. The manual will also be available here. He is also considering adding a searchable online database of aliases and how-to articles. This may also become the preferred way to submit bug reports, suggestions, or any other contribution to the project.

http://greens.ml.org/about/elvis/online-help/elvis.html
: This is one location on the web where elvis' manual is available online. Be sure to check the version number!

http://www.phys.columbia.edu/~flame/vi.htm
    This is a Web page for elvis, maintained by JaeSub Hong. It has some screen shots, macros, and syntax-coloring definitions. It also has the online manual.

http://www.thomer.com/thomer/vi/vi.html
    "The VI lover's home page." This contains links to a wide variety of vi documentation, and practically all vi clones. *A very good resource!* This is a new location; previously it was located at http://www.cs.vu.nl/~tmgil/vi.html.

http://www.math.fu-berlin.de/~guckes/vi/features.html
    Feature comparison between different VI implementations, plus a few other editors. Originally it just listed which options are supported by which clones, but it has matured a bit since then. The maintainer is most familiar with VIM, so there is a bias towards that.

http://www.bsyse.wsu.edu/~rnelson/editors/editors.htm
    A compendium of text editors. It describes all kinds of text editors, not just vi clones. It lists the features and supported platforms for each editor.

http://darren.hiebert.com/ctags/
    Home page for Darren Hiebert's **exuberant ctags** program. It supports all features of elvis' ctags, except for the "ln" attribute, and the "-r" flag for generating a "refs" file. Exuberant ctags has a smarter parsing algorithm, which causes it to generate fewer bogus tags (i.e., tags for things that really shouldn't have tags). It also adds the ability to generate tags for "enum" values and a few other useful things.

http://wafu.netgate.net/tama/unix/global.html
    Home page for Shigio Yamaguchi's **global** function reference utility. You can make elvis use it instead of the normal built-in tag searcher by giving elvis the following command:

```
:se tagprg="global -t $1"
```

    Once you have run **global**'s gtags utility, you can do some powerful things such as search for each place where a function (e.g., m_front()) is called, like this:

```
:ta -r m_front
```

http://www.cs.berkeley.edu/~amc/Par/
    Home page for Adam Costello's **par** program, which is a much more sophisticated text formatter than the fmt program distributed with elvis. It even does a very good job of reformatting comments! Sadly, it is one of those rare programs that doesn't handle tab characters correctly.

http://sourceware.cygnus.com/cygwin/
    This is the home page for the CygWin tools -- ports of GNU utilities to Microsoft Windows, by Cygnus. If you're looking for more Unix utilities to go with elvis, this is the place.

ftp://alf.uib.no/pub/vi
    An archive site containing many macro packages and other information about vi. Nearly all of it should apply equally well to elvis.

ftp://ftp.oce.nl/pub/vim
http://www.clark.net/pub/dickey/vile/vile.html
http://www.sleepycat.com/vi/
http://www.snafu.de/~ramo/WinViEn.htm
http://ourworld.compuserve.com/homepages/fwiarda/software.htm
http://www.emi.net/~jjensen/java/javi.html
    These are the home pages for some other vi clones: vim, vile, nvi, WinVi, pvic, and javi respectively. The last one is interesting because it is written in Java!

ftp://ftp.funet.fi/pub/doc/posix/p1003.2a/d8/5.10
ftp://ftp.funet.fi/pub/doc/posix/p1003.2a/d8/5.35
    These are old drafts of the POSIX standards for ex and vi, respectively. These URLs might not be valid very long.

http://www.de.freebsd.org/de/doc/usd/12.vi/paper.html
    This is the official BSD documentation for vi.

http://alumni.caltech.edu/~dank/nansi/
    This is a home page for the NANSI.SYS and NNANSI.SYS drivers, which accelerate the screen updates under MS-DOS.

ftp.leo.org:/pub/comp/os/os2/leo/gnu/emx+gcc/
ftp-os2.cdrom.com:/pub/os2/emx09d/
ftp-os2.nmsu.edu:/pub/os2/dev/emx/v0.9d/
    These are all places where you can find the EMX.DLL library for OS/2. The first one is the "home" site, and the others are popular OS/2 archives. You need EMX only if you want to run the "termcap" (elvisemx.exe) or "x11" (elvisx11.exe) versions of elvis under OS/2.

http://set.gmd.de/~veit/os2/xf86os2.html
    This is the home page for the OS/2 version of XFree86. You need this (in addition to EMX) if you want to run the graphical X11 version of elvis.

16.2 Using elvis as a Web browser

NOTE: The following information doesn't apply to the MS-DOS version of elvis, because that version doesn't support the **ftp** and **http** protocols. But for Win32, Unix, and OS/2...

You can use elvis as a light-weight Web browser. Surfing with elvis isn't as much fun as surfing with a multimedia-capable browser such as Netscape or MSIE, but elvis does have some advantages: it starts up much faster, it feels like vi, and you can edit whatever you download.

There's no special trick to loading a Web page. Just give a URL where elvis expects a filename, and elvis will read the Web page via the Internet. You can follow links in Web pages just as you do in elvis' online manual. HTML pages are displayed in the "html" display mode, and anything else uses the "hex" or "normal" display mode by default. All data is fetched in binary so data files aren't mangled; however, this also means that newlines aren't converted, which may make non-HTML text files look ugly.

Elvis has built-in support for the HTTP and FTP protocols. Other protocols may be indirectly supported, via an HTTP proxy as indicated by the elvis.net configuration file.

By default, FTP access is anonymous. However, if you give a file name which starts with "/~/" then elvis will attempt to login to the FTP server using you own account, as described in the elvis.ftp or ~/.netrc configuration file. You can also use "/~username/" to use some other user account listed in .netrc. For example, "ftp://localhost/directory/file" uses anonymous FTP, but "ftp://localhost/~/directory/file" uses your own account.

Elvis can write via FTP as well as read; see the Internet chapter.

Elvis also doesn't support inline graphic images, but that isn't as big of a problem as you might think. If you download an image, elvis will simply load it into a buffer and then display that buffer in the "hex" display mode. You can then write that buffer's contents to a file, or in Unix you can send it directly to an image viewer via a command such as ":w !xv -".

The easiest way to save an image (or any other buffer) to a local file is via the command ":w (dirfile(filename))". In fact, you might want to add the following lines to your ~/.exrc file to make the **(F2)** key save the current buffer to a file, and the **(F3)** key send it to the xv image viewer:

```
        map #2 :w (dirfile(filename))^M
        map #3 :w !xv -^M
```

To make images easier to fetch, any <IMG SRC=*url*> tag which isn't already part of a hypertext link will be interpreted as a link to the SRC *url*. This allows you to download an image by moving the cursor onto it and hitting the **(Enter)** key.

Elvis doesn't support frames either, so a similar trick was used for <FRAME SRC=*url*> tags. Elvis displays the name of each frame; those names serve as links to the contents of the frame.

Because elvis is primarily an editor, not a Web browser, I deliberately made the "html" display mode rather picky, so that any questionable entities in your own HTML documents will call attention to themselves. When you're using elvis to browse other peoples' documents, though, this can be annoying, so I modified it slightly to be more forgiving when you're viewing read-only documents. (All Web pages are read-only.)

And you already know that elvis' support for the <TABLE> macros is very poor, right? If you encounter a Web page which looks really ugly in elvis, you can bet it uses tables.

Elvis doesn't support forms, or secure connections. Well, elvis can display mock-ups of forms; they just don't work. They probably never will. You have to draw the line somewhere.

Elvis doesn't always choose the best display mode for HTML pages. It uses "html" if the file name ends with ".html" or ".htm", or if the document's text begins with "<!", "<H", or "<h". For all other documents, it uses the "normal" or "hex" display mode by default. If elvis chooses the wrong display mode, you can use the :display command to switch to a different display mode.

The command ":e foo" will always load the local file "foo" from your current directory. This is true of all commands which normally act on files -- unless you give a complete URL, elvis assumes it should work with local files. However, while in the "html" display mode, the command ":ta foo" will use the same protocol, site, and directory as the page you're already viewing, because that's how the "html" display mode interprets tags.

16.3 How to debug macros

There are two ways to create a macro in elvis: You can either assign a series of commands to a keystroke (or series of keystrokes) via the :map command, or you can store a series of commands in a cut buffer and execute them via the visual @x command. You will often use a combination of techniques, in which :map macro constructs a customized @x macro and runs it.

(Aliases are a separate issue, discussed later in this chapter. The information in this section does not apply to aliases. Elvis does not yet offer any special tools for debugging aliases.)

Elvis has several features that make debugging macros much easier. For example, you can create a window which continuously displays the contents of a given cut buffer, such as "m, via the command:

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

or, more concisely:

```
        :("m)sp
```

Note: The cut buffer must exist before you can display it. Cut buffers are created the first time anything is yanked into them.

The maptrace option allows you to trace the execution of macros. You can either allow it to run through the macro, or wait for a keypress before each mapped command character. You can also use the :break and :unbreak commands to set or clear a breakpoint on a given :map macro. Breakpoints cause the maptrace option to switch from "run" to "step" when that macro is expanded.

The maplog option can be used to log the trace information to a buffer named "Elvis map log". The idea here is that you will give the command...

```
        :se mt=r mlog=r
```

... (or its full-length form, :set maptrace=run maplog=reset) before starting the macro, and then when the macro fails you can give the command...

```
        :(Elvis map log)split
```

... to see what it was doing shortly before the failure. Note that the maplog option has no effect if maptrace is "off".

Warning: Elvis has a single keystroke queue which is shared by all windows. Because of this, while elvis is running a macro in one window you can't switch to another window and type in commands. Even if the GUI allows you to switch windows without using the keyboard, doing so will simply force the macro to continue execution in the new window. *So don't switch windows while a macro is running!*

Here's a debugging methodology that works for me:
1) Begin by loading the macro package and a test file.
2) Give the command ":se mt=r mlog=r", and run the macro.
3) If the macro fails, give the command ":(Elvis map log)split" to find out what commands executed immediately before the failure. In particular look for a :map macro that was expanded shortly before the failure.
4) Set a breakpoint on that macro with ":break *macrokey*".
5) Turn off logging, via ":se mlog=o".
6) Reload the test file.
7) Execute the macro again. When the macro with the breakpoint is encountered, elvis will switch to single-step mode. Step slowly through the next few instructions, looking for one which does something unexpected.

If your macro reveals a bug in elvis, please let me know! My email address is kirkenda@cs.pdx.edu. Please tell me which version of elvis you're using, as reported by the :version command.

16.4 Running your compiler from within elvis

Elvis can parse most compilers' error messages. When it parses an error message, elvis loads the faulty file, moves the cursor to the line where the error was reported, and shows the descriptive portion of the error message on the bottom row of the window. You can step through all reported errors very quickly, making changes along the way.

Usually, you will invoke your compiler or "make" utility via the :cc or :make commands. The only difference between these commands is that :cc invokes the program named by the ccprg option, and :make uses the makeprg option.

Both of those options' values are evaluated using the simpler expression syntax, with $1 set to any extra command-line parameters, and $2 set to the current file name.

You can also read error messages from some other program with the command ":errlist !*program*", or read them from a file with the command ":errlist *filename*".

I often invoke elvis via the command "elvis +make" so elvis will attempt to compile the program, and move the cursor to the first error (if there are any errors).

All of the compiler's output text is collected into a buffer named "Elvis error list". If you wish, you can view this list in a separate window via this command:

```
:(Elvis error list)split
```

Here's how elvis parses each line of compiler output: Starting from the left, it divides the line into "words", which are defined as a series of letters, digits, and/or certain punctuation characters.

If the word is the name of an existing directory, then elvis remembers that directory name. In later lines, elvis will allow file names to be given relative to that directory, in addition to the current directory. This particular feature is intended to work with the directory lines generated by the GNU version of the "make" program.

If the word looks like a number, and no line number has been seen yet, then the word is taken to be a line number. If the word is the name of an existing, writable file (or any existing file if the anyerror option is set) in either the current directory or the directory remembered from a previous line as described above, then the word is taken to be a file name. Other words are ignored.

When elvis has found both a file name and a line number, then it skips over any whitespace or punctuation immediately following them, and uses the remainder of the line as the error's description.

If elvis fails to find a file name/line number pair, then it skips that whole line of compiler output.

Immediately after collecting compiler output, elvis moves the cursor to the source of the first error. After that, you can use :errlist (with no arguments) or the visual * command to step through each following error.

Each time elvis collects a new set of error messages, it remembers how many lines are in each buffer. Later, when you insert or delete lines to correct an error, elvis can compare the current number of lines to original number of lines, and adjust the reported line numbers accordingly.

Here's something that may be useful for PERL programmers. PERL's error messages follow two distinct formats:

> *description* **in file** *file* **at line** *line*
> *description* **at** *file* **line** *line*

Neither of these looks like any recognizable compiler error message format; consequently, elvis can't parse PERL's error messages directly. But here's a way around that. The following PERL program is a filter that reformats PERL's error messages to look like normal compiler error messages.

```
#!/usr/bin/perl
$\ = "\n";
while (<>) {
    chop;
    s/(.*) in file ([^ ]*) at line (\d*).*/$2($3): $1/;
    s/(.*) at ([^ ]*) line (\d*)\.$/$2($3): $1/;
    print;
}
```

To use this script, store it in a file named "perlerr" and turn on the file's "execute" permissions, and then set elvis' ccprg option as follows:

```
For CSH:          :set ccprg="perl -c ($1?$1:$2) |& perlerr"
Other shells:     :set ccprg="perl -c ($1?$1:$2) 2>&1 | perlerr"
```

NOTE: You can't simply cut&paste the above perl script into a file, because it contains some HTML code which PERL wouldn't understand. (The diamond is written as "&lt;&gt;".) A better way is to visually select those lines via the shift-V command, and then use the :wascii alias to save the formatted text to a file as plain ASCII text. You'll still need to edit the text file to remove leading whitespace and possibly some blank lines, but that's pretty easy.

16.5 Internationalization

Elvis can be configured to translate its messages into different languages, and to use different symbol sets. These things are accomplished via the elvis.msg file and :digraph command, respectively.

Elvis locates the elvis.msg file during initialization. Ordinarily it searches through each directory named in the ELVISPATH environment variable. However, if there is an environment variable named LC_ALL, LC_MESSAGES, or LANG (listed in order or precedence) which is set to a non-null value, then elvis will look for elvis.msg first in a subdirectory whose name matches the environment variable's value. For example, if LC_ALL is unset, LC_MESSAGES=german, and ELVISPATH=~/.elvis:/usr/local/lib/elvis, then elvis would try to load its messages from...

1) ~/.elvis/german/elvis.msg
2) ~/.elvis/elvis.msg
3) /usr/local/lib/elvis/german/elvis.msg

4) /usr/local/lib/elvis/elvis.msg

The digraph table tells elvis which pairs of ASCII characters can be combined to form a single non-ASCII character. This table is configured via the :digraph command. To enter a digraph, type **<Ctrl-K>** and then the two ASCII characters. Elvis will store the corresponding non-ASCII character instead of the two ASCII characters. See the Input Mode chapter for more information.

The digraph table affects more than just keyboard input. It also affects "html" mode, and character type classifications.

Digraphs are used by the "html" display mode to translate character entities into characters. For example, when elvis encounters &ntilde; in an HTML document, it tries to find a digraph which combines 'n' with '~'. If there is such a digraph, elvis will use it to display an 'ñ'; if not, then elvis will display a plain 'n' character.

The digraph table affects the character classes, too. This, in turn, affects the definition of a "word", as used by the visual w command, among others. A non-ascii character is treated as an uppercase letter if, according to the digraph table, it is the result of combining an ASCII uppercase letter with either a punctuation character or a second uppercase letter. A similar rule holds for lowercase letters.

Also, elvis tries to find uppercase/lowercase pairs through the digraph table. This is used for case conversions, as performed by the visual ~ command, or the \U metacharacter in the :s/old/new command.

There is no way to specify a sorting order. This means, in particular, that the regular expression /[a-z]/ will only match the ASCII lowercase letters, not the non-ASCII ones. However, the regular expression /[[:lower:]]/ *will* match all lowercase letters including the non-ASCII ones.

The default elvis.ini file tries to load digraphs by executing either elvis.pc8 for MS-DOS, OS/2, or text-mode Win32, or elvis.lat for any other operating system.

The "win32" version of the "termcap" user interface has a codepage option which determines which symbol set is used for console output. If you change codepage, you should also adjust your digraph table.

16.6 Aliases

Aliases provide a simple way to add a new name for an existing ex command, or series of ex commands.

The syntax of elvis' :alias command is intended to resemble that of the csh Unix shell. The simplest example of an alias is...

```
:alias save w
```

... which would allow you to write your file out by running ":save", as an alternative to the standard ":w". If you pass any arguments to ":save" then they'll be appended to the ":w" command. For example, ":save foo" would be interpreted as ":w foo".

Here's another example. On Unix systems, this will make ":ls" display a directory listing.

```
:alias ls !!ls -C
```

Note that the above example requires *two* exclamation marks. This is because the "!" character is special in aliases -- elvis' aliases allow you to use special symbols to indicate where arguments belong in the command text, and all of those symbols begin with a "!" character. When you invoke the alias, all of the symbols are replaced by argument values before the command text is executed. Here is a complete list of the replacement symbols:

SYMBOL	REPLACED BY
!<	first address line, if any addresses given
!>	last address line, if any addresses given
!%	address range, if any addresses given
!?	"!" if the alias is invoked with a "!" suffix
!*	the entire argument string except for "!" suffix
!^	the first word from the argument string
!$	the last word from the argument string
!*n*	the *n*th word (where *n* is a single digit)
!!	a single, literal "!" character

Using any of the **!***, **!^**, **!$**, or **!1** through **!9** symbols in the command string has the side-effect of disabling the normal behavior of appending the arguments to the command. Or to phrase that another way: If the command text doesn't explicitly say what to do with arguments, then elvis will assume it should simply append them.

The other symbols, such as **!%** and **!?**, have no such default behavior. If your macro is going to use addresses or a "!" suffix, then you must explicitly include **!%** or **!?** (respectively) in the command string.

Here's a simple alias for playing around with these:

```
:alias show echo !!<=!< !!%=!% !!?=!? !!*=!* !!^=!^ !!2=!2 !!$=!$
```

Here's a more sophisticated version of the ":save" alias. This version allows you to use ":save!" as an alias for ":w!".

```
:alias save w!?
```

Here's a macro that converts a range of lines to uppercase. If invoked without any addresses, it will change only the current line, because that's the default for the :s command.

```
:alias upper !%s/.*/\U&/
```

You can define multi-line aliases by enclosing the lines in curly braces. The following example uses this technique to make a slightly smarter version of the "save" alias:

```
:alias save {
 "Write a file, but only if it has been modified
 if modified
 then w!? !*
}
```

Note that the first line of the alias's body is a comment. (Comments start with a " character.) This is a good idea because when the :alias command is invoked with no arguments, it lists the names and first lines of all aliases. Putting a descriptive comment in the first line allows you to see what each alias does simply by examining that list.

If a multi-line alias is going to use arguments, then it must include **!***, **!^**, **!$**, or **!***n* symbols. Elvis does *not*, by default, append arguments to the end of a multi-line alias; it only does that for single-line aliases.

An alias can have the same name as a built-in command, but aliases can't be recursive. Together, these two rules mean that you can use an alias to change the behavior of a built-in command. For example, the following alias makes the :w command perform an RCS checkout operation if you don't already have write permission for a file. The "w" command inside the command text refers to the normal :write command since it isn't allowed to be a recursive call to the "w" alias.

```
:alias w {
 "Write a file, checking it out first if necessary
 if readonly && "!%!?!*" == ""
 then !!co -l %
 then w!!
 else !%w!? !*
}
```

You can optionally insert a ':' character between the '!' and the second character of any of these symbols. This has no effect; it is allowed simply to remain a little closer to CSH's alias syntax.

You can also optionally insert a '\' character between the '!' and the second character. This *does* have an effect: It causes a backslash to be inserted before any characters which would otherwise receive special treatment if they appeared in a regular expression. Specifically, it will always insert a backslash before '\', '/', '^', '$', '.', '[', or '*'. Note that this is *not* sensitive to the magic option; in effect, it assumes that magic is always set. Also, it never inserts a backslash before a '?' character even if it is used in a regular expression which is delimited by '?' characters. The following "find" alias will search for literal text:

```
:alias find /!\*
```

In addition, you can optionally specify a default value for an argument, by placing the value in parentheses between the '!' and the second character. Here's an example which acts like echo, except that if you don't tell it what to echo then it will echo "Howdy!":

```
:alias greet echo !(Howdy!)*
```

If necessary, you can insert both a backslash and a parenthesized default value. The backslash quoting will be applied to the given argument value, but not the parenthesized default value. Here's a variation of the "find" alias which searches for literal text, or if you don't specify any text to find then it searches for a { character at the front of a line.

```
:alias find /!(^{)\*
```

Some vi commands are implemented via ex commands. If you create an alias with the same name as a built-in ex command, then the corresponding visual command will be affected. For example, the ZZ visual command runs :x, so the following alias would break the ZZ command...

```
:alias x echo Winners never quit, and quitters never win
```

16.6.1 Some example aliases

The distribution comes with some handy aliases in a file named lib/elvis.ali. I suggest you look at them. The simple ones should give you some ideas of how to structure your own aliases, and the complex examples will give you a feel of what can be accomplished.

Remember that the names of aliases must be spelled out in full; you can't abbreviate alias names the way can for built-ins. Also, you can display the definition of any alias by running ":alias *aliasname*".

These examples are intended to be useful as well as instructive. They are loaded automatically when elvis starts. The aliases in that file include:

:lf [*directory*]
List the contents of the current directory, or of a named directory. On Unix systems this works by invoking "ls -CF" on the arguments; on other systems it invokes "dir/w".

:pwd
Display the name of the current directory.

:howto[**!**] *word* [*word2*]
Load the "How To" appendix in a separate window, and search for a topic containing the given word or words. The words should be typed in lowercase. If you want to search all lines (not just topic lines) then run :howto! (with a ! suffix).

:**kwic** *word*
Build a table showing all occurences of *word* in the online manual. This macro depends on the "grep" program to do the actual searching; if your system lacks "grep" then this won't work correctly. (Windows users should consider using the CygWin tools.)

:**man** [*section*] *topic*
Unix only. Read a man-page and display it in a new window.

:**save** [*filename*]
Write this file, but only if it has been modified.

:[*range*]**w**[**!**] [*filename*]
Like the normal :w command, except that if you try to write a whole file back over itself, and that file is readonly, then the :w alias will attempt to perform an RCS "checkout" operation on that file by running "!co -l *filename*".

:**courier** [*size*]
X11 only. Select the courier fonts of the given size. If no size is specified, then it uses the default size, which is 18-point.

:**copying**
Display the license.

:**cbsave** *filename*
:**cbload** *filename*
Save the cut buffers ("a through "z) to a file, or load them from that file.

:**cbshow** *cutbufs*
Display the contents of one or more cutbuffers. The *cutbufs* string is interpreted as a list of one-character cutbuffer names. If you invoke :cbshow without any arguments then it will show all cutbuffers.

:**config**
Report the configuration of your copy of elvis. I.e., list the features which were enabled when elvis was compiled.

:**customize** *filename*
Create a personal copy of one of elvis' configuration files (unless you already have a personal copy of it), and start editing it. A typical example would be ":customize elvis.syn" to edit the syntax coloring rules.

:[*range*]**left**
:[*range*]**right**
:[*range*]**center**
For each line in the given range (or only the current line if no addresses are given), adjust the indentation so that its text is moved to the left, right, or center of the line.

:[*range*]**rot13**
Perform ROT13 encryption/decryption on the given range of lines,

or only the current line if no addresses are given. ROT13 is a simple character-substitution code in which characters from the front half of the alphabet are exchanged with the corresponding characters from the back half of the alphabet. It is sometimes used as a courtesy when posting offensive jokes to the rec.humor newsgroup, along with an unencrypted warning such as "The following may be offensive to *whomever*. Decode using ROT13 at your own risk."

:[*range*]**wascii** *file*
: Write WYSIWYG formatted text out to a file as plain ASCII text. This is intended to make it easy to save ex scripts (or other source code) that are embedded within HTML pages such as this manual. It works by temporarily setting lptype to "dumb", and then writing the text via :lp.

:[*range*]**makehtml**
: Convert plain text into HTML source. By default it acts on the whole edit buffer, but you can also tell it to convert only the lines in a given range. Uppercase lines are assumed to be headings. Lines which start with a number or asterisk are assumed to be list items. Indented lines are assumed to be preformatted text. It also attempts to create a link for each URL or email address in the text.

:[*range*]**cfmt**
: Adjust the line breaks in a C or C++ comment block.

:[*range*]**align** [*symbol*]
: Align any = signs (or other given symbol) in a range of lines. This is useful for making macro definitions in a Makefile look pretty. This is the most complex example.

:match
: If the cursor (in visual mode) is on an "if", "then", "case", or "do" keyword then this moves the cursor to the matching "fi", "else", "esac", or "done" keyword, respectively. It can also do the reverse.

  This macro is very easy to modify to support different words. Near the top of the macro, two variables named **x** and **y** are set to slash-delimited lists of the beginning and ending words. All you need to do is change those lists.

16.7 How to make elvis run faster

This section describes some ways you can "tune" elvis to run faster. None of these suggestions require recompiling elvis.

For example, my computer (AMD K6-200) can run 10 generations of the "life" macros in 41 seconds with the default configuration. If I invoke elvis with a reduced block size (-b1024 on the command line) and an increased cache (:set blkcache=200 blkhash=300), it can run 10 generations in just 24 seconds.

The blksize option
Elvis uses fixed-size blocks to store buffers. The block size is chosen when the session file is created. The default is 2048 bytes (or 1024 bytes for MS-DOS), but you can override that by invoking elvis with a **-b***blksize* flag. The size must be a power of two, between 512 and 8192.

The blksize option indicates the current block size. This is a read-only option; once elvis has started, (and hence has already created the session file) it is too late to request a different block size.

Generally, smaller blocks are better if your CPU is slow or you're only editing small files. Larger blocks are better if your disk is slow (e.g., the temporary file is stored on a remote disk, accessed via a network) or you're editing large files.

The block cache
In the interest of speed, elvis has its own cache of blocks from the session file. The blkcache option tells elvis how many blocks to store in the cache. You can change this value at any time. If elvis requires more cached blocks for a given editing operation than the blkcache allows, then elvis ignores blkcache and loads the required blocks into the expanded cache; you can't make blkcache too small. The default blkcache is 20 (except for MS-DOS, where it is 10), and the upper bound is 200 blocks. In MS-DOS, setting blkcache too high can cause elvis to crash.

The blkhit and blkmiss options count the number of cache hits and misses, so you can compute the efficiency of the cache as follows:

```
        :calc (bh*100/(bh+bm))"%"
```

You'll probably find that 98% or more of the block requests are being satisfied from the cache. However, since each miss takes thousands of times longer to complete than a hit, each fraction of a percent means a lot.

In addition to the blocks themselves, the cache contains a hash table which allows elvis to quickly determine whether a block is in the cache or not. If you increase the size of the cache, then you'll probably want to increase the size of the table as well. The table size is controlled by the blkhash option. Ideally, it should be set to a prime number somewhat larger than blkcache... or simply the largest possible value, since hash table entries are small. The default is 61, and the upper bound is 300.

Syncing
Elvis has a sync option which, if set, causes elvis to force all dirty blocks out to disk occasionally. This is a *very* slow operation, so the sync option is normally turned off. If elvis seems to be running exceptionally slowly, then you might want to verify that sync is off. You can check it by giving the command ":set sync?".

Temporary files

The session file, and any other temporary files, should be stored on a local hard disk. Storing them on a network drive will slow elvis down a lot.

The session option indicates where the current session file is stored, and the sessionpath option indicates which directories elvis looked through when deciding where to put the session file. These are read-only options, since it is too late to choose a new location for the session file after the session file is already created. If you need to force elvis to store its session files in a different directory, you should set the SESSIONPATH environment variable to a list of acceptable directories. Elvis will use the first directory from that list which exists and is writable.

The directory option tells elvis where to store other temporary files, such as those used for piping text through external programs. You can change its value at any time. (Note: the real vi also has a directory option, but only allows you to change it during initialization.)

Screen updates
Options which add information to the bottom row of the window, such as ruler and showmode, can slow down screen updates. If speed is critical, you should turn those options off.

The optimize option attempts to eliminate superfluous screen updates while a macro is executing. It is normally on, but you may want to verify that. Some animation macros force it off.

Elvis also has an animation option which, if optimize is on, causes elvis to bypass some of the animation steps. The default value is 3. If animations look choppy then try reducing it to 1. Or set it to 10 or so for faster animation.

The exrefresh option controls whether elvis should refresh the screen after each line output by an ex command, or wait until the command completes and then update the screen once. It is normally off, so the screen is only updated once.

Interestingly, the syntax-coloring display mode seems to run about as fast as the normal display mode. One possible exception would be if you're running elvis over a slow modem connection then the extra escape sequences required for sending color information may slow down screen updates.

If you have long lines, then elvis may run somewhat faster when the wrap option is set. This is because elvis always formats entire lines, even if only part of the line is visible on the screen, and the "nowrap" display style shows more long lines (one per row) than the "wrap" display style.

Input
The pollfrequency option indicates how often elvis checks for an abort request from the user. (Abort requests are usually given by typing ^C or by clicking on a window while a macro or other time-consuming command is running). Smaller numbers give a quicker

response to an abort request, but will cause the command itself to run slower.

The keytime option indicates how long elvis should wait after receiving an Esc character to distinguish between the <Esc> key, and some other function key which begins with an Esc. Longer times are more reliable, especially when you’re running over a network. But shorter times allow a quicker response to the <Esc> key.

MS-DOS-specific tips

The single biggest improvement you can make is to switch from the 16-bit MS-DOS version to the 32-bit text-mode Win32 version. It only runs under Windows95 or WindowsNT, not MS-DOS, but you can make it run in full-screen mode which feels like MS-DOS. And it is *much* faster, because I really had to mangle the MS-DOS version of elvis to make it fit in the lower 640k.

The fancier ANSI drivers such as NANSI.SYS also help. They allow elvis to redraw the screen very quickly. The URLS section of this chapter tells you where you can find NANSI.SYS on the Internet.

Installing smartdrv.exe can be a *big* help. Storing temporary files on a RAM disk (in extended/expanded memory please!) can also help.

Elvis also has a compile-time option, controlled by the FEATURE_RAM declaration in config.h, which allows elvis to store its buffers in EMS/XMS memory instead of a file. This makes elvis run *much* faster, but it has some disadvantages. If elvis crashes, there will be no way to recover the contents of the edit buffers. Also the Microsoft functions for accessing EMS/XMS memory are very bulky, and also require a fairly large buffer; so if you enable FEATURE_RAM, then you must disable most of the other FEATURE_XXXX features in config.h, and even then elvis may run out of memory occasionally. If you enable FEATURE_RAM, then to actually use that feature, you must invoke elvis with "-f ram" on the command line.

If you often use "-b 2048" or "-f ram", then you might consider setting up a batch file which runs them. For example, you could create a elvis.bat file containing...

```
        elvis.exe -b 2048 -f ram %1 %2 %3 %4 %5 %6 %7 %8 %9
```