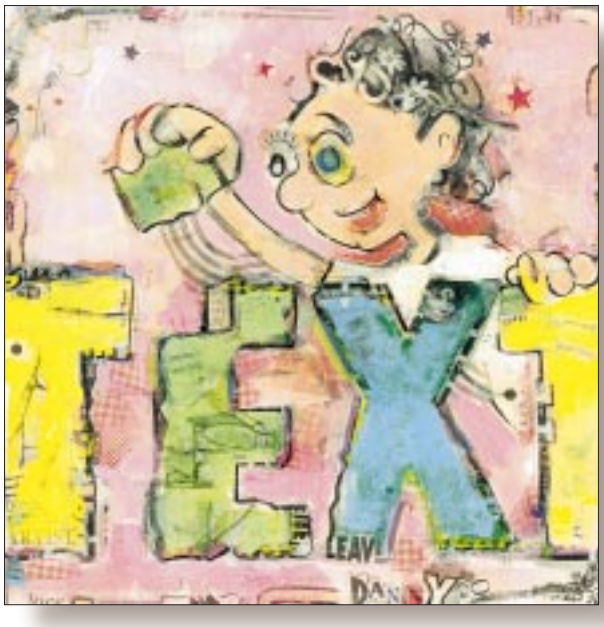
by Jeffreys Copeland and Haemer



We Use vi to Edit Web Pages

Jeffrey Copeland (copeland@alumni.caltech.edu) *is a member of the technical staff at QMS' R&D group in Boulder, CO. He's been a software consultant to the Hugo award administrators for several years. He spends his spare time raising children and cats.*

Jeffrey S. Haemer (jsh@canary.com) *now works for QMS, too, and is having a great time. Before he worked for QMS, he operated his own consulting firm, and did a lot of other things, like everyone else in the software industry.*

We're going to show you how to turn `vi` into a WYSIWYG HTML editor. WYSIWYG, for those of you who aren't veterans of the editor or word processor wars, stands for "what you see is what you get." We think it was our friend Mark Kampe who originally used the old Flip Wilson line to distinguish between embedded-directive and keyboard-command word processors.

Our friend, Tom Schneider, at the National Cancer Institute, has been hard at play again. Longtime readers will recall that Tom is a fellow who keeps prodding us into thinking about problems that end up as columns.

Typically, the way this works is that Tom has an idea, writes a script to implement it that doesn't quite work, sends it to us with a request for help, and we get sucked into working on it. Of course, most of Tom's scripts work; if they don't have problems, he doesn't send them to us. What keeps turning the things he sends us into columns is that they are *interesting* ideas.

The most recent of these columns evolved from Tom's desire for a watchdog that would monitor files for changes. Beginning with Tom's first-cut shell script, we wrote a replacement script in Perl that solved his original problems and showed him how to refine it into an interpreter for a small, but real, programming language, `atchange`, all in about 50 lines of code.

A simple invocation such as

```
atchange foo date
```

will print the date every time the file `foo` changes. This is useful, for example, if you're monitoring a slow process that only provides occasional output.

At the other end of the scale, Listing 1 shows an example `atchange` program that watches a variety of files and takes different actions when each file changes.

Tom has posted a copy of our original column at `http://www-lmmb.ncifcrf.gov/~toms/atchange.paper/atchangepaper.html`. If you're in a browsing mood, we encourage you to go to Tom's home page and look at some of the other stuff he's doing.

Listing 1

```
#!/usr/local/bin/atchange
#
# Here's a program for atchange

       HELLO="hello world"  # set a variable
       echo $PS1

/tmp/hello  echo $HELLO  # all one script

       datefn() {      # define a function
          echo the date: $(date)
       }

/tmp/date datefn
       echo -n "$PWD$"

       counter=0

/tmp/counter      # commands can span multiple lines
       echo $counter
       let counter=counter+1

       CLEARSTR=$(clear)

/tmp/iterator
       echo $CLEARSTR
       let iterator=iterator+1
       echo $iterator | tee /tmp/iterator

/tmp/zero_counter
       let counter=0
       touch /tmp/counter
```

Tricks with Netscape

Last month, Tom dropped us a note that builds on `atchange`. We're not the only folks Tom talks to, of course. Recently, Stephen Eglen, at the University of Sussex, pointed out to Tom that he was able to send instructions to a running invocation of Netscape from the command line. For example, `netscape -remote 'openURL(http://www.qms.com/)'` will cause the Netscape you're currently running to go to the QMS home page. (If you're not running Netscape, the command will just print an error message and fail.) The folks at Netscape explain other commands that you can give Netscape from the command line at `http://home.netscape.com/newsref/std/x-remote.html`.

This seemed perfect. Tom reasoned that he could tell `atchange` to watch an HTML file he was editing in `vi`; each time it changed, `atchange` could tell Netscape to redisplay it. Listing 2 shows the code to do exactly that.

Exegesis

Let's go through the code in Listing 2 line by line. Line 1 is the "shebang" (`#!`) line that tells the system what interpreter to use. We use `bash`, but another POSIX-conforming shell, like `ksh`, should work fine, too. Line 2 is our RCS ID. Yes, we really keep our shell scripts under source code control.

Lines 3 to 7 are comments, including a comment about how to install the code if you're putting it onto a new system. We try to make our code as portable as we practically can, but there are often problems with portability around the edges, and documenting them in the code helps whoever's trying to get it to run.

Line 8 is safety netting that we put in reflexively. In theory, this line is there to guard against Trojan horses. In practice, it's mostly valuable as a guard against unusual individual `PATH` settings. In other words, it helps us catch instances where we depend on particular versions of programs being in our `PATH` settings in preference to the standard versions.

Lines 9 to 27 are a shell function that prints out a usage message. If this were a Perl script, we'd just have a simple usage message and construct a full-blown man page, integrated with the code, using Perl's "pod" facility, for more complete documentation. (See the perlpod(1) man page for details.) For tiny shell scripts like this, however, we tend to be lazier. In this case, we've created two kinds of usage messages: a typical UNIX one-line synopsis of the proper invocation, and a longer help message, which isn't really a man page, but will do until we write one.

If you're not used to shell parameter expansion, line 11 will look mysterious. This statement trims the directory information from `$0`, the name of the script, and puts the result in `$ARGV0`.

We could have used `basename` to do the trimming, but parameter expansion lets you do the same job in the shell itself, without the cost of spawning a new process.

We challenge the reader who wants to learn more about parameter expansion to try to figure out why this statement

```
: ${PERL5LIB=/usr/local/lib/perl5}
```

sets the value of `PERL5LIB` to `/usr/local/lib/perl5` if, and only if, `PERL5LIB` isn't already assigned a value. (N.B.: The initial colon is important. Don't leave it out!) We use this trick to provide default values inside shell scripts that can be overridden by environment variables.

By the way, as you can see from lines 11, 12 and 19, we *never* hard-wire the name of the program into the program itself. It's too easy to rename the executable but forget to change it within the code. As an example, this program, which we now call `eh`, had at least three different names while we were developing it.

Note also, in lines 12 and 14, that we make sure to send error messages to standard error, not standard out. This is the sort of care most programmers take with their C programs but often neglect to take with their shell scripts.

Lines 28 through 50 encapsulate a template HTML page. We've spent a fair amount of time tinkering with our template, and we expect we'll continue to do so as our tastes change. If

Listing 2

```
1  #!/usr/local/bin/bash
2  # $Id: eh,v 1.2 1997/03/26 16:10:42 jeff Exp $

3  # edit an html file while you watch the results in Netscape
4  #     Netscape will refresh every time you write the file

5  # INSTALLATION:
6  #     - fix the shebang line and the PATH
7  #     - make sure atchange is installed

8  PATH=/usr/local/bin:/bin:/usr/bin

9  # Print usage message and exit
10 usage() {
11      ARGV0=${0##*/}
12      echo "usage: $ARGV0 [-l|-h|-help] filename|filename.html" 1>&2
13      test "$1" = "long" || exit 1

14      cat <<-__EOD__ 1>&2

15      Netscape will display the indicated html file
16      and the editor will be invoked on it simultaneously.
17      Whenever you write out the file in the editor,
18      Netscape will update.

19      $ARGV0 will create a template if the named file doesn't exist,
20      and will add an .html extension to the filename
21      if you don't type it.

22      For further information, see
23      http://www-lmmb.ncifcrf.gov/~toms/atchange.html
24      or write to Tom Schneider, <toms@ncifcrf.gov>.
25 __EOD__
26      exit 1
27 }

28 # Make a template html document
29 html_template() {
30      cat <<-__EOD__
31              <html>
32              <head><title>
33                      FIXthisTITLE
34              </title></head>

35              <body
36                      bgcolor="#EEFFFA"
37                      text="#000000"
38                      link="#CC0000"
39                      alink="#FF3300"
40                      vlink="#000055"
41              >

42              <center>
43              <h1>
44                      FIXthisHEADING
45              </h1>
```

your taste in Web pages is different from ours, here's where you tinker.

For colors (lines 36 to 40), we follow David Siegel's recommendations, which he discusses in detail at `http://www.dsiegel.com/tips/wonk2/background.html`.

Line 51 brings us to the main body of the program. We begin by checking for proper invocation in lines 51 to 54, and then use parameter expansion again to put the file name into a standard form. After lines 55 to 60, `$html` will hold the absolute path of a file whose name ends in `.html`.

Line 62 checks to see whether the file already exists. If not, it creates the file, using the template we specified earlier. The trick of using `||` and `&&` to implement simple conditionals is a little confusing at first, but it's a common idiom in shell scripts. We could have written lines 57 to 60 as the single command

```
test "${html#/}" = "$html" &&
  html=$PWD/$html
```

but we left it as four to present a contrast of the two forms for you.

Line 62 tells the Netscape browser you're running to display the current version of the file `$html`. We follow this immediately with an `atchange` command that will tell Netscape to redisplay this file whenever it changes.

Finally, we start up the editor, once again using parameter expansion. This time, we start up `vi` unless the variable `$EDITOR` is set in the environment, in which case we use the editor it names. This lets you use `ed` as your editor if you really want to. (This line is really for the benefit of our boss, Steve, who decided he didn't like `vi` *or* `emacs`, and wrote his own screen editor: `se`–Steve's Editor.)

Lines 66 and 67 provide us with a gracious exit. `kill 0` kills off the `atchange`, and `wait` waits for it to finish before the script exits, to avoid creating a zombie.

Last, but not least, line 24 illustrates a small but useful design principle: Always point complaints at someone else.

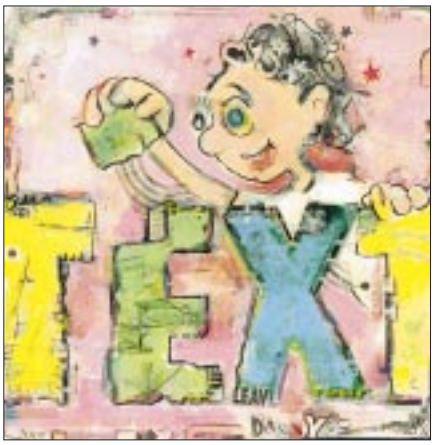
Listing 2 (continued)

```
46            </center>

47            PUTsomethingHERE

48            </body></html>
49  __EOD__
50  }

51  case $1 in
52     -l|-h|-help)  shift; usage long ;;
53  esac
54  test $# -eq 1 || usage

55  # Add .html if necessary
56  html=${1%%.html}.html
57  if test "${html#/}" = "$html"
58  then
59     html=$PWD/$html
60  fi

61  test -f $html || html_template > $html

62  netscape -noraise -remote "openFile($html)"
63  atchange $html "netscape -noraise -remote 'reload'" &

64  # Finally, invoke the editor:
65  ${EDITOR:-vi} $html
66  kill 0
67  wait
```

Sows' Ears and Silk Purses

OK, so we now have something that lets us use `vi` as a WYSIWYG HTML editor: So what?

First, we've thrown in a lot of little tips, and we hope even those readers who don't want to use `eh` itself will have picked up a trick or two. If you've been reading this column for a while, you'll know that's our normal approach: create something useful, but make getting there half the fun.



Second, we confess that we use `vi` all the time. Any flavor of `vi`–`vi`, `nvi`, `elvis`, `vim`, `viper`–you name it.

Yes, we've used a lot of other editors. There are even tasks for which we routinely use (gasp!) `emacs`–especially debugging with `gdb`–if only to keep our control, alt and escape keys from getting lonely.

Still, for garden-variety editing we always seem to come back to `vi`.

We can rationalize this by saying that it's a standard (POSIX 1003.2), or by arguing that it's small and well integrated into the rest of the UNIX tool set, or by pointing out how beautifully ergonomic its cursor-motion sequences are. Really, we suspect that it's mostly because we've been using it for so long that its commands are wired into our fingers.

This preference is pervasive. Our `.profile` files contain the line `set -o vi`, to let our fingers use `vi` commands to search and edit our shell command histories, and our `.emacs` files contain the lines `(require 'viper)` and `(setq vip-always t)`, so that we can use `vi` commands inside `gnus`, the `emacs` newsreader, when replying to Usenet postings.

We write our columns using `vi` and `troff`. Naturally, therefore, we want to continue to use `vi`, even when we're building Web sites.

We don't even think our preference is unusual. We are, for example, willing to bet that most of the columnists for this magazine also write everything from columns to email with a simple text editor like `vi`, rather than WYSIWYG versions of some damnfool what-you-see-is-not-necessarily-what-you-want word processor.

Putting our money where our mouths are, we bet our editors a nickel, hard cash. Lisa and Lisa: Put up, or shut up.

(Actually, Lisa and Lisa already know better, and declined to take the bet. Their exact response was, "What kind of Rubes do you take us for?" Any other takers?)

A Side Note

A couple of months ago, ("Counting on the Net," RS/*Magazine*, February 1997, Page 29), we discussed errors in some order-of-magnitude numbers people have been throwing around while discussing IPv6 and competing Internet addressing schemes. We did this by deriving things like the number of protons in the Earth from stuff we learned in high school geometry and chemistry classes. Since then we have read Bruce Schneier's *Applied Cryptography*, Second Edition, Wiley, 1996, ISBN 0-471-12845-7 (hard cover) or 0-471-11709-9 (paperback). We discovered that for comparison purposes, Schneier provides a whole list of large numbers in Table 1.1. His 10^{51} atoms in the Earth nicely compares with our order-of-magnitude calculation of $3x10^{52}$ protons in the Earth. Check out the table for some other interesting numbers.

Until next month, happy trails.