The journal of Apple technology.

Volume Number: 19 (2003)
Issue Number: 1
Column Tag: Section 7

Still More Perl

Munging Mail and Media...

by Rich Morin

Perl's "whipitupitude" is legendary. This column looks at a couple of small scripts I've recently been "whipping up", showing how Perl can work in and around more formal OSX tools. One script, fmmf, Finds Monster Mail Files; I use it to keep track of mailing list (and other) mail files which may be getting out of hand. The other script, cfwc.d, is a daemon (background process) which helps me operate an experimental webcam.

Finding MOnster Mail Files

I'm on quite a few mailing lists and I don't always get to the associated mailboxes regularly to keep them under control. I'm also trying to track the efficacy of my spam filtering system (based on SpamAssassin and Eudora), which drops suspected spam into one of several mailboxes, depending on its numeric spam rating, etc. I have written a short script which helps me keep on top of these issues.

The mainline code, below, is quite simple. Using finddepth, from the File::Find module (available on the CPAN; cpan.perl.org), it performs a depth-first examination of my email folder. The callback function, wanted, is invoked for each node (e.g., file, directory) in the tree. Using the lists produced by this traversal, the remaining code prints out the results for spam and miscellaneous email, sorting each list in a case-insensitive manner.

```
#!/usr/bin/env perl
#
# fmmf - find monster mail files
#
# Written by Rich Morin, CFCL, 2002.11
use File::Find;
$monster = 2000000;
{
  $eu = '/Users/rdm/Mail/Eudora Folder';
  finddepth(\&wanted, "$eu/Mail Folder");
  for $line (sort {lc($a) cmp lc($b)} (@spam)) {
    print $line;
  }
  print "\n";
  for $line (sort {lc($a) cmp lc($b)} (@misc)) {
    print $line;
  }
}
```

The tricky parts of this script, such as they are, lie in the "wanted" callback function. As it traverses the tree, finddepth changes the "current directory" and sets $_ to the relative name of the node. This makes it easy to skip over items that aren't files and Eudora's "table of contents" (*.toc) files.

```
sub wanted {
  return unless (-f $_);
  return if ($_ =~ m|\.toc$|);
```

For the next part, however, we need the "full path name" of the node. Getting this from a handy helper method, we can strip off the first part of the path and test the remainder in assorted ways. Perl's regular expressions are very useful for this sort of name handling.

```
  $path = $File::Find::name;
  $path =~ s|^.*/Eudora Folder/Mail Folder/||;
  return if ($path =~ m|_Inactive/ Save/|);
```

After picking up the size of the file (in bytes), the script opens each mailbox in the "spam" area and counts the number of "From: headers (i.e., messages). Eudora uses carriage returns (rather than the conventional BSD newlines) for line termination, but setting Perl's $/ (input record separator) variable handles that quite easily. The strings containing the formatted output are pushed into a list, for use by the mainline code.

```
  $size = -s $_;
  if ($path =~ m|!Spam|) {
    open(MBOX, $_) or die "can't open mailbox($_)";
    $/ = "\r";
    $fcnt = 0;
    while (defined($line = <MBOX>)) {
      $fcnt++ if ($line =~ m|^From:|) ;
    }
    close(MBOX);
    push(@spam, sprintf("%-35s  %9d  %4d\n",
      $path, $size, $fcnt));
    return;
  }
```

The code for miscellaneous mailboxes is comparatively simple. After ensuring that the mailbox is large enough to qualify as a "monster", it formats and saves the output lines. Perl's "x" operator comes in handy for creating a "quick and dirty" histogram.

```
  return if ($size < $monster);
  $isiz = int($size/$monster);
  push(@misc, sprintf("%-35s  %9d  %s\n",
    $path, $size, '*' x $isiz));
}
```

This sort of "personalized" script is quite common in BSD circles. Clearly, it isn't suitable for use by others, as is, but it is short and simple enough that it can easily be customized to meet the needs of different users. Here is some sample output, from my own system:

```
!Spam/?? Junk (Eudora)                9041     5
!Spam/?? Junk (SA 1)                 39192     6
!Spam/?? Junk (SA 2)                 11467     2
```

```
!Spam/?? Junk (SA 3)                420538    60
_Lists/DocBook                     3231686  *
_Lists/FreeBSD/FreeBSD-Ports       6431902  ***
_Lists/FreeBSD/FreeBSD-Questions   2666962  *
...
```

A WebCam Daemon

I recently started playing with an iBOT, a FireWire-based camera made by Orange Micro

(www.orangemicro.com). My initial goal was to create a simple "security camera" app that would display a set of recent images on a web page.

After downloading the OSX driver for the iBOT, I started looking around for image capture software. One package, EvoCam (www.evological.com), captures images, based on elapsed time and/or software-based motion detection. It can also upload the image files (via FTP) to a web server and/or save numbered copies on the local disk.

Unfortunately, this wasn't exactly what I wanted. The FTP upload feature simply refreshed the same file; turning this into a time history would be tricky. The numbered image files would do, however, if I could get them over to the web server. All told, it was a good start on what I wanted. All I needed to do was create a little plumbing...

The first part of the plumbing had to do with getting the files from my desktop Mac onto the (FreeBSD-based) local web server. FreeBSD provides NFS, but getting OSX to mount the provided volumes can be quite a trial. Fortunately, Marcel Bresink's NFS Manager (www.bresink.de/osx/NFSManager.html) eases the pain considerably.

Once I got the files sifting into a directory on the web server, I merely had to rename them (for convenience) and build up a web page to display a selected subset. The following script, while still a "work in progress", accomplishes these tasks quite handily.

```
#!/usr/bin/env perl
#
# cfwc.d - Canta Forda WebCam Daemon
#
# Written by Rich Morin, CFCL, 2002.11
$imgs = '/.../iBOT';   # adjust to taste...
$html = '/.../cfwc';   # adjust to taste...
{
  for (;;) {
```

As mentioned above, EvoCam generates a unique name (e.g., 123456789.jpg) for each image file. In writing these to the NFS-mounted FreeBSD machine, OSX also generates a companion file (e.g., ._123456789.jpg) for the resource fork. The code below creates a new name for the image file, based on the file's modification time, and discards the companion file.

```
    # Clean out incoming directory.
    opendir(IN, "$imgs/incoming")
      or die "can't open $imgs/incoming";
    @in = grep(!/^\./, readdir(IN));
    chomp(@in);
    closedir(IN);
    for $in (sort(@in)) {
      @stat = stat("$imgs/incoming/$in");
```

```
      $mtime = $stat[9];
      ($sec, $min, $hour, $mday, $mon, $year,
       $wday, $yday, $isdst) = localtime($mtime);
      $out = sprintf("%d.%02d%02d.%02d%02d%02d.jpg",
        $year+1900, $mon+1, $mday, $hour, $min, $sec);
      rename("$imgs/incoming/$in",
             "$imgs/i.queue/$out");
      unlink("$imgs/incoming/._$in");
    }
```

Perl's approach to reading directories is rather messy, but it isn't all that difficult. The code below gets a list of filenames, discarding any that don't match the desired format, and sorts them. Because the names were crafted with this in mind, the list is now in chronological order.

```
    # Get list of images to display.
    opendir(IN, "$imgs/i.queue")
      or die "can't open $imgs/i.queue";
    @in  = sort(grep(/^\d{4}\.\d{4}\.\d{6}\.jpg$/,
                     readdir(IN)));
    chomp(@in);
    closedir(IN);
```

Using Perl's "slice" syntax, we grab the last (i.e., most recent) nine file names.

```
    @show = @in[-9 .. -1];
```

Now we start generating a web page. The META tag tells the user's browser to refresh the page every 15 seconds. I am rather compulsive about formatting the HTML; the web browser doesn't care, but it sure makes debugging less painful for humans!

```
    # Make up a new web page.
    open(OUT, ">$html/index.temp")
      or die "can't open index.temp";
    print OUT <<EOT;
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Refresh" content="15">
    <TITLE>Canta Forda WebCam</TITLE>
  </HEAD>
  <BODY>
    <TABLE>
EOT
```

The code below generates a 3x3 table of images, each followed by a centered label. I could have used the file names (e.g., 2002.1129.2039.jpg) as labels, but that would have been a bit ugly. Why not parse the names and reformat the values into a more readable format?

Note the multi-line regular expression that is used to break up the file name. When REs get long and complex, breaking them up in this manner can make them much easier to follow.

```
    $cnt = 0;
    for ($i=0; $i<9; $i+=3) {
```

```
      print OUT "      <TR>\n";
      for ($j=0; $j<3; $j++) {
        print OUT "        <TD>\n";
        $k = $i + $j;
        $tmp1 = $show[$k];
        $tmp1 =~
          m|^(\d{4})\.            # (YYYY).
           (\d\d)(\d\d)\.         # (MM)(DD).
           (\d\d)(\d\d)(\d\d)\.   # (HH)(MM)(SS).
           jpg$|x;                # jpg
        $tmp2 = sprintf("%s/%s/%s at %s:%s:%s",
                        $1, $2, $3, $4, $5, $6);
        print OUT "          <CENTER>\n";
        print OUT "            ",
                  "<IMG SRC=\"iq/$tmp1\"><BR>\n";
        print OUT "            $tmp2\n";
        print OUT "          </CENTER>\n";
        $cnt++;
        print OUT "        </TD>\n";
      }
      print OUT "      </TR>\n";
    }
```

Finally, we push out the last of the HTML, close the file and (Oh, yes!) move it into place for Apache to find. Then, after a second's repose, we go back up and do the whole exercise again.

```
    print OUT <<EOT;
    </TABLE>
  </BODY>
<HTML>
EOT
    close(OUT);
    rename("$html/index.temp",
           "$html/index.html");
    sleep(1);
  }
}
```

Lessons Learned

As we all know, the Mac and BSD universes aren't a perfect fit. Perl is a very good "glue language", however, allowing us to deal smoothly with issues such as line termination, extra (e.g., resource fork) files, etc.

Similarly, there are a wealth of useful apps which can perform small tasks, fill in gaps between operating systems, and generally make our lives easier. If a $20 shareware package can save me hours of frustration, the purchase decision is a no-brainer.

Unfortunately, some issues are still difficult to resolve. For instance, although it's easy to scan a Eudora mail file for header lines, editing Eudora mailboxes would be far trickier. Aside from file locking problems, there is the small issue of the (binary, undocumented) format of the TOC files. In short, choose your challenges carefully...

Rich Morin has been using computers since 1970, Unix since 1983, and Mac-based Unix since 1986 (when he helped Apple create A/UX 1.0). When he isn't writing this column, Rich runs Prime Time Freeware (www.ptf.com), a publisher of books and CD-ROMs for the Free and Open Source software community. Feel free to write to Rich at rdm@ptf.com.