Perl 101 (part 8) – Putting It To The Test

By Vikram Vaswani and Harish Kamath

Table of Contents

Reach Out And Touch Someone

Last time out, we discussed how Perl can be used to write simple CGI applications that reside on the server and do useful things like receiving form data. This time, we're going to take it a little further, by showing you how to make Perl do some really useful things.

Over the next few pages, we're going to show you how to track the number of visitors your page receives with a simple counter, store their comments on your Web development efforts in a basic guestbook, and write a simple mailer that can email the contents of a feedback form to the site webmaster. So keep reading − this is your chance to get to know your site visitors better!

Adding Things Up

The first tool that we're going to discuss today is the counter. The basic purpose of a Web site counter is to record the number of people who have visited a particular page.

There are four simple steps involved in building a counter:

Step One: Create a text file on the server, which will store the number of visitors.

Step Two: When someone visits the site, this file is accessed, the number stored within it is obtained and incremented by one, and then displayed to the visitor.

Step Three: The new number is written back to the file.

Step Four: Go back to Step Two.

Here's the code:

```
#!/usr/bin/perl # counter.cgi – count visitors to page # open
a file handle to read the contents of the file "counter.dat" #
this file contains the number of visitors before the # current
user. # make sure that you have permission to write to this
file open (COUNT, "counter.dat"); # assign the value that is
stored in the file handle in a temporary variable $counter =
<COUNT>; close (COUNT); # open the file handle to write the
new number back to the file open (COUNT, "> counter.dat"); #
increment the counter variable by one $counter += 1; # write
the new number print COUNT "$counter"; close (COUNT); # now
display the HTML code print "Content-Type: text/html\n\n"; #
print the new counter value print "<html>\n<body>\n<h1>Your
are visitor number $counter.</h1>\n</body>\n</html>";
```

And here's the output:

```
Content-Type: text/html <html> <body> <h1>Your are visitor
number 30.</h1> </body> </html>
```

You can easily extend the functionality of this counter to store the IP address, the browser type, the referrer URL and other visitor statistics by making use of the environment variables we showed you last time.but we're going to leave that to you.

Visitors Welcome!

Next up, a guest book. We're going to create a simple guest book that asks the user to enter three parameters: name, email address and comment.

In order to make things simple for you, we've included the basic HTML code for the form as well.all you need to do is cut, copy and paste into your favourite HTML editor. Good service, huh?

```
<html> <head> <basefont face=Arial> <title>Guest Book</title>
</head> <body> <form action="submit.cgi" method="post">
<center> <h2>Your Space In My Space</H2> <table width="600"
bgcolor="#D6D6D6" cellpadding="10" cellspacing="5"> <tr> <td
width="300" align="right">Name</td> <td width="300"
align="left"><input type="text" name="name" size="25"
maxlength="25"></td> </tr> <tr align="center"> <td width="300"
align="right">Email</td> <td width="300" align="left"><input
type="text" name="email" size="25" ></td> </tr> <tr> <td
align="right" width="300">Comments</td> <td align="left"
width="300"><textarea name="comments" cols="25" rows="3"
wrap="virtual"></textarea></td> </tr> <tr> <td colspan=2
align="center" width="600"><input type="submit" value="Sign My
Guestbook"></td> </tr> </table> </div> </form> </body> </html>
```

As you can see, this form asks the user to enter three values: Name, Email address and Comments. The ACTION of the form is a CGI script that will accept the values entered into the form and store them in a text file.

The Code...

Here's the other half of the puzzle − the CGI script that takes care of the actual storage of form data.

```
#! /usr/bin/perl # submit.cgi − accepts guestbook data and
writes to file # define a variable that accepts a value from
the form $in; # assign values to the variable depending on
form METHOD if ($ENV{'REQUEST_METHOD'} eq "GET") { $in =
$ENV{'QUERY_STRING'}; } else { $in = <STDIN>; } # fix
URL−encoded strings $in =~ s/\+/ /g; # all variables are
passed to the script as name−value pairs separated by &# split
the input string on the basis of &@detail = split (/$in); #
display data entered by user again print "Content−Type:
text/html\n\n"; print "<html><body>"; print "<center>"; print
"<table cellspacing=5 cellpadding=5 width=600
bgcolor=#D6D6D6>"; print "<tr><td align=center colspan=2
width=600><font face=Verdana size=2>Thank you for entering the
following details in the guestbook.</font></td></tr>\n"; #
each name−value pair is stored as an element of an array # now
take each element of the array and split to form a hash # on
the basis of the = symbol # using the "foreach" loop, we split
each element of the array into a "temporary" hash foreach
$details(@detail) { %details = split (/=/, $details); # now
extract the name−value pair from the temporary hash to display
to the user # some forward thinking here: # since values need
to be stored in a text file, create a variable called $entry #
and differentiate the different elements of each guestbook
entry by the # symbol # this comes in useful later, wait and
see! while (($name, $value) = each %details) { print "<tr><td
align=right width=300><font face=Verdana size=2>Your
$name:</font></td><td align=left width=300><font face=Verdana
size=2> $value</font></td></tr>\n"; $entry .= $value . "#"; }
} # end the display with a link that allows the user to view
other guestbook entries print "<tr><td align=center colspan=2
width=600><font face=Verdana size=1>Click <a href=
view.cgi>here</a> to view other
entries.</font></td></tr></table>"; print "</body></html>"; #
make things simple by ensuring that every entry in the file is
on a single line # by terminating the entry with a newline
$entry = $entry."\n"; # this is where the file write actually
happens # remember to open the file in "append" mode to avoid
losing previous data # make sure that you have permission to
write to this file open (GBOOK, ">> guestbook.txt"); print
GBOOK "$entry"; close (GBOOK); # whew!
```

Now, how about an explanation?

...And The Explanation

Let's start from the top. We've first defined a variable, $in, that accepts a value from the form. As explained in our previous example, all the form variables are passed to the script as name−value pairs. However, in this case, we are passing more than one name−value pair, separated from each other with an ampersand.

Therefore, it becomes necessary to split the input string against the &to get the individual name−value pairs. Each name−value pair is stored as an element of an array. When we begin printing the data entered (for verification), it becomes necessary to again split each element of the array into individual "names" and "values" against the = symbol. Splitting things up this way also makes it simple to write the different items to a text file.

Using the "foreach" loop, we split each element of the array, as described above, into a temporary hash variable. We can then extract the name−value pair from the temporary hash and display it to the user.

Since we have to also store the values in a text file, we've created a variable named $entry, formatted it in such a way that it contains all three values entered by the user (separated by a # symbol), and dumped it into a text file. Remember to open the file in "append" mode!

Going Backwards

So that takes care of data entry. But how about the link that allows the user to read previous guestbook entries? Well, here's the code for the CGI script that reads the file and displays all previous entries to the user.

```
#! /usr/bin/perl # view.cgi – display guestbook entries # open
the file and read contents into an array open (GBOOK,
"guestbook.txt"); @entries = <GBOOK>; close (GBOOK); print
"Content-Type: text/html\n\n"; print "<html><body>"; print
"<center>"; print "<table cellspacing=5 cellpadding=5
width=600 >"; print "<tr><td align=center width=600><font
face=Verdana size=2>View other comments.</font></td></tr>\n";
# at this point, each entry in the file is stored as a single
element of the array foreach $entry(@entries) { # use chomp()
to delete the newline character from the end of each line
chomp $entry; # split each array element against the #
delimiter into a new temporary array # now, the first element
of the temp array contains the name # the second, the email
address and the third, the comment @singles = split (/#/,
$entry); # display it print "<tr><td width=600>"; print
"<table cellspacing=5 cellpadding=0 width=600
bgcolor=#D6D6D6><tr ><td align=right width=300 ><font
face=Verdana size=2>Name:</font></td><td align=left
width=300><font face=Verdana
size=2>$singles[0]</font></td></tr>\n"; print "<tr><td
align=right width=300><font face=Verdana size=2>Email
address:</font></td><td align=left width=300><font
face=Verdana size=2>$singles[1]</font></td></tr>\n"; print
"<tr><td align=right width=300 valign=top><font face=Verdana
size=2>Comments:</font></td><td align=left width=300><font
face=Verdana size=2>$singles[2]</font></td></tr></table>\n";
print "</td></tr>"; } print "</body></html>";
```

Now, when it's time to display the contents of the guestbook, we do things in reverse: first read the file into an array, split each element against the # delimiter we added earlier, and display it in a neatly formatted table. Simple, huh?

Fortune Cookies

Perl also allows you to run external commands, and display the output of those commands on your Web page. Consider the following simple fortune cookie generator, which uses the "fortune" program to give you a random quote each time you reload the page.

```
#!/usr/bin/perl # fortune.cgi − get a random quote # get a
quote − change your path appropriately $quote =
`/usr/games/fortune`; # print it in a page print
"Content−Type: text/html\n\n"; print <<EOF; <html> <head>
<basefont face=Arial> </head> <body> And your quote is: <br>
$quote </body> </html> EOF
```

In this case, we've executed a command by enclosing it in single quotes, and sent the output to the variable $quote. Next, we've used Perl to output an HTML page containing the quote − this page is the one you'll see when you visit the site through your browser. Each time you refresh it, you'll see a new quote.

Note the slightly different manner in which we've structured the print() statement here. The << marker indicates to Perl that what comes next is a multi−line block of text, and is to be printed as is right up to the marker (the marker in this case is the string "EOF"). In Perl−lingo, this is known as a "here document", and it comes in very handy when you need to output a chunk of HTML code.

You Have Mail!

And on to our final example of the day − a simple form mailer. Let's assume that you have a feedback form which looks like this:

```
<html> <head> <style type=text/css> td {font−family: Arial}
</style> </head> <body> <h2>So Who Are You, Anyway?</h2> <form
action="mailform.cgi" method=post> <table border="0"
cellspacing="5" cellpadding="0"> <tr> <td>Name</td> <td><input
type=text name=who size=30></td> </tr> <tr> <td>Email
address</td> <td><input type=text name=email size=30></td>
</tr> <tr> <td valign="top">Address</td> <td><textarea
name="address" cols="30" rows="5"></textarea></td> </tr> <tr>
<td>Age</td> <td><input type=text name=age size=2></td> </tr>
<tr> <td align=center colspan=2><input type=submit
value=Send!><input type=reset></td> </tr> </table> </form>
</body> </html>
```

Now, you need to have the contents of this form emailed to you every time someone submits it. Here's the Perl script that you'll need:

```
#! /usr/bin/perl # mailform.cgi − email form contents to
webmaster # define a variable that accepts a value from the
form $in; # assign values to the variable depending on form
METHOD if ($ENV{'REQUEST_METHOD'} eq "GET") { $in =
$ENV{'QUERY_STRING'}; } else { $in = <STDIN>; } # fix
URL−encoded strings $in =~ s/\+/ /g; # all variables are
passed to the script as name−value pairs separated by &# split
the input string on the basis of &@detail = split (/$in); #
open mail pipe open (MAIL,"|/usr/sbin/sendmail −t"); print
MAIL "To: <webmaster\@yoursite.com>\n"; print MAIL "From:
Feedback Form Mailer\n"; print MAIL "Subject: Feedback on your
site\n\n"; print MAIL "Here is the result of your feedback
form.\n\n"; foreach $details(@detail) { %details = split (/=/,
$details); while (($name, $value) = each %details) { print
MAIL "$name: $value\n"; } } close MAIL; print "Content−Type:
text/html\n\n"; print "<html><body><center><font face=Arial
size=+1>Thank you for your
feedback!</font></center></body></html>";
```

If you take a close look, you'll see that this script is very similar to the one we used when writing the guestbook. Here too, we've accepted form data as a single string, split it on the basis of the ,and then further split it against the = separator.

Next, we've opened a file handle − except that we haven't actually opened a file, but a UNIX "pipe", which allows you to "pipe" data to a UNIX command. In this case, the command is the sendmail program, which is

used to deliver email. sendmail needs a few basic headers − the To:, From: and Subject: fields, which we've provided, followed by each name−value pair in the body of the message. Once all the pairs are exhausted, the handle is closed and the mail is sent out.

And here's the sample mail that you'll see:

```
To: webmaster@yoursite.com From: Feedback.Form.Mailer Subject:
Feedback on your site Here is the result of your feedback
form. who: johndoe email: johndoe@cyberspace.com address: the
web age: 28
```

And that's about it for this time. We hope you enjoyed this series of tutorials − write in and tell us what you'd like to see next. And till next time, stay healthy!