

Beginning Perl Programming (X52.9543.001)**Handout 7: July 6, 2000****From Homework 5 ...**

Here's code to "massage data into a concise report" for the web server error log:

```
open (TRYIT, "| uniq -c | sort -nr | head -5");
print TRYIT @errorlist;
close (TRYIT);
```

- use of pipes
- what is the shortcoming of this technique?

The REAL story on what m// returns ...

```
#!/App/Perl/bin/perl
# mtest.pl
# This script demonstrates the return values of the m// match operator in
# a scalar and a list context.

$s = "A String";

print "1: ", scalar ($s =~ m/A(..)(ring)/), "\n"; # Successful match
print "2: ", scalar ($s =~ m/A(..)(rang)/), "\n"; # Failed match

print "3: ", join ("|", $s =~ m/A(..)(ring)/), "\n"; # Successful match
print "4: ", join ("|", $s =~ m/A(..)(rang)/), "\n"; # Failed match

# Here's a useful idiom for the list-valued return of a match:

if ( ($word1, $word2) = ($s =~ m/(\w+)\s+(\w+)/) ) {
    print "5: $word1|$word2\n";
}
```

Output of mtest.pl under MKS Toolkit 6.1 under Win98 using ActivePerl

```
1: 1
2:
3: St|ring
4:
5: A|String
```

Isolating the message portion of a error_log line

```
$line =~ s/\\[\\[.*\\]//;

if ( /^\\[(.+?)\\] \\[(\\w+)\\] \\[(.+?)\\] (.+)$/ ) { ...
```

logHit.pl

Here's a routine straight out of the Web/perl package used by a large company. It takes information about the client making the Web request and updates hit log and DB Hash files.

```
# Logs relevant data for a "hit" to a log file.
# On problems, we call dieCgi, unless the -noDie option is set to nonzero.
# Returns undef on success, or an error string on failure
# (if the -noDie option is set to nonzero)
#
# Options:
# -noDie      If set to non-zero, this routine return the text of any
#             error message.
#
# -noMetrics  Only write the hit.log file, not the metrics DB_HASH files.
#
# Example:
# my ($errorText) = LogHit ("-noDie" => 1);
# if ($errorText) { die $errorText; }

sub LogHit {
    my (%options) = @_ ;

    my ($remHost) = $ENV{'REMOTE_HOST'};
    my ($remAddr) = $ENV{'REMOTE_ADDR'};
    my ($remUser) = $ENV{'REMOTE_USER'};

    my ($fname) = "../log/hit.log";
    my ($hitHandle) = new FileHandle;

    if (! open ($hitHandle, ">> " . $fname)) {
        my ($complaint) = "cannot open $fname: $!";
        if ($options{'-noDie'}) {
            return $complaint;
        } else {
            &dieCgi ($complaint);
        }
    }

    # Grab a writer-block on the file

    if (! &Sync:::writer ($hitHandle)) {
        my ($complaint) = "error synchronizing on $fname: $!";
        if ($options{'-noDie'}) {
            return $complaint;
        } else {
            &dieCgi ($complaint);
        }
    }

    # We used to fetch the time via:
    #   = localtime ($^T);
    # HOWEVER, that got the time at the startup of the script, which is a
    # bad idea in the case of a long-lived Interface Process.

    my ($sec, $min, $hour, $mday, $mon, $year, $yday, $wday)
        = localtime (time);

    print $hitHandle "URL: ", &MyURL, "\n";
}
```

```

if ($remUser) {
    printf $hitHandle
        "Date: %02d/%02d/%04d @ %02d:%02d:%02d %15s %s <%s>\n",
        $mon + 1, $mday, $year + 1900, $hour, $min, $sec,
        $remAddr, $remHost, $remUser;
} else {
    printf $hitHandle
        "Date: %02d/%02d/%04d @ %02d:%02d:%02d %15s %s\n",
        $mon + 1, $mday, $year + 1900, $hour, $min, $sec,
        $remAddr, $remHost;
}

unless ($options{'-noMetrics'}) {

    # Log it to the usage database by incrementing the count of
    # hits for this host.
    # A NOTE on locking: We keep our lock on the hit log file
    # established above, and use it to coordinate access
    # to the db_usage database.

    my ($dbdir) = "../db_usage/";
    my ($access) = O_RDWR | O_CREAT;
    my ($mode) = 0666;

    my (%dateCountTable);
    my ($todayDate) = sprintf ("%04d-%02d-%02d",
        $year + 1900, $mon+1, $mday);
    my ($dateCountDbHandle) = tie (%dateCountTable, "DB_File",
        $dbdir . "usage_date_count.hsh",
        $access, $mode, $DB_HASH);
    $dateCountTable{$todayDate}++;
    untie %dateCountTable;

    my (%dayCountTable);
    my ($dayCountDbHandle) = tie (%dayCountTable, "DB_File",
        $dbdir . "usage_day_count.hsh",
        $access, $mode, $DB_HASH);
    $dayCountTable{$yday}++;
    untie %dayCountTable;

    my (%hostCountTable);
    my ($hostCountDbHandle) = tie (%hostCountTable, "DB_File",
        $dbdir . "usage_host_count.hsh",
        $access, $mode, $DB_HASH);
    $hostCountTable{$remHost}++;
    untie %hostCountTable;

    my (%hourCountTable);
    my ($hourCountDbHandle) = tie (%hourCountTable, "DB_File",
        $dbdir . "usage_hour_count.hsh",
        $access, $mode, $DB_HASH);
    my ($formatHour) = sprintf ("%02d", $hour);
    $hourCountTable{$formatHour}++;
    untie %hourCountTable;

    if (!$remAddr) { $remAddr = "Unknown"; }
    my (%ipCountTable);
    my ($ipCountDbHandle) = tie (%ipCountTable, "DB_File",
        $dbdir . "usage_ip_count.hsh",

```

```

        $access, $mode, $DB_HASH);
$ipCountTable{$remAddr}++;
untie %ipCountTable;

if (!$remUser) { $remUser = "Unknown"; }
my (%userCountTable);
my ($userCountDbHandle) = tie (%userCountTable, "DB_File",
    $dbdir . "usage_user_count.hsh",
    $access, $mode, $DB_HASH);
$userCountTable{$remUser}++;
untie %userCountTable;
}

# Free our writer's block

if (! &Sync::free ($hitHandle)) {
    my ($complaint) = "error freeing lock on $fname: $!";
    if ($options{'-noDie'}) {
        return $complaint;
    } else {
        &dieCgi ($complaint);
    }
}

if (! close ($hitHandle)) {
    my ($complaint) = "cannot close $fname: $!";
    if ($options{'-noDie'}) {
        return $complaint;
    } else {
        &dieCgi ($complaint);
    }
}

return undef;
}

```

Improved csv_parse

```
#!/App/Perl/bin/perl -w
#
# csv_parse2.pl
#
# Perl script to demonstrate parsing of CSV (comma-separated value) lines
# (c) Copyright 2000 by Clint Goss, All Rights Reserved.

# VERY brief summary of CSV file format rules:
# 1. Records are newline-terminated.
# 2. Fields are separated by commas.
# 3. Field values containing commas or double quote characters
#    are enclosed in double quotes.
# 4. Double quote characters inside of a field value are duplicated.

# Example: the field value: Phil, Brent, Bill, Mickey, "Bobbie", and Jerry
# would appear in a CSV as: "Phil, Brent, Bill, Mickey, ""Bobbie"", and Jerry"

use CSV;

# Loop to read lines of input and parse them:

while (<>) {
    my (@fieldValues) = &fieldsOfCSVRecord ($_);

    print "\nInput line:\n$_";
    print "\nFields:\n", join ("\n", @fieldValues), "\n";
}
exit;

# Take a record from a CSV file and return the field values as a list.

sub fieldsOfCSVRecord {
    my ($line) = @_;
    chomp ($line);

    my ($csv) = new Text::CSV;
    if (! $csv->parse($line)) { die "Can't parse $line\n"; }

    my (@fields) = $csv->fields();
    return @fields;
}
```

NAME

Text::CSV - comma-separated values manipulation routines

SYNOPSIS

```
use Text::CSV;

$version = Text::CSV->version();    # get the module version

$csv = Text::CSV->new();            # create a new object

$status = $csv->combine(@columns);  # combine columns into a string
$line = $csv->string();             # get the combined string

$status = $csv->parse($line);       # parse a CSV string into fields
@columns = $csv->fields();          # get the parsed fields

$status = $csv->status();           # get the most recent status
$bad_argument = $csv->error_input(); # get the most recent bad argument
```

DESCRIPTION

Text::CSV provides facilities for the composition and decomposition of comma-separated values. An instance of the Text::CSV class can combine fields into a CSV string and parse a CSV string into fields.

FUNCTIONS**version**

```
$version = Text::CSV->version();
```

This function may be called as a class or an object method. It returns the current module version.

new

```
$csv = Text::CSV->new();
```

This function may be called as a class or an object method. It returns a reference to a newly created Text::CSV object.

combine

```
$status = $csv->combine(@columns);
```

This object function constructs a CSV string from the arguments, returning success or failure. Failure can result from lack of arguments or an argument containing an invalid character. Upon success, `string()` can be called to retrieve the resultant CSV string. Upon failure, the value returned by `string()` is undefined and `error_input()` can be called to retrieve an invalid argument.

string

```
$line = $csv->string();
```

This object function returns the input to `parse()` or the resultant CSV string of `combine()`, whichever was called more recently.

parse

```
$status = $csv->parse($line);
```

This object function decomposes a CSV string into fields, returning success or failure. Failure can result from a lack of argument or the given CSV string is improperly formatted. Upon success, `fields()` can be called to retrieve the decomposed fields. Upon failure, the value returned by `fields()` is undefined and `error_input()` can be called to retrieve the invalid argument.

fields

```
@columns = $csv->fields();
```

This object function returns the input to `combine()` or the resultant decomposed fields of `parse()`, whichever was called more recently.

status

```
$status = $csv->status();
```

This object function returns success (or failure) of `combine()` or `parse()`, whichever was called more recently.

error_input

```
$bad_argument = $csv->error_input();
```

This object function returns the erroneous argument (if it exists) of `combine()` or `parse()`, whichever was called more recently.

EXAMPLE

```
require Text::CSV;

my $csv = Text::CSV->new;

my $column = '';
my $sample_input_string = "I said, \"Hi!\"\",Yes,\"\",2.34,,\"1.09\"";
if ($csv->parse($sample_input_string)) {
    my @field = $csv->fields;
    my $count = 0;
    for $column (@field) {
        print ++$count, " => ", $column, "\n";
    }
    print "\n";
} else {
    my $err = $csv->error_input;
    print "parse() failed on argument: ", $err, "\n";
}

my @sample_input_fields = ('You said, "Hello!"',
                           5.67,
                           'Surely',
                           '',
                           '3.14159');
if ($csv->combine(@sample_input_fields)) {
    my $string = $csv->string;
    print $string, "\n";
} else {
    my $err = $csv->error_input;
    print "combine() failed on argument: ", $err, "\n";
}
```

CAVEATS

This module is based upon a working definition of CSV format which may not be the most general.

- 1 Allowable characters within a CSV field include 0x09 (tab) and the inclusive range of 0x20 (space) through 0x7E (tilde).
- 2 A field within CSV may be surrounded by double-quotes.
- 3 A field within CSV must be surrounded by double-quotes to contain a comma.
- 4 A field within CSV must be surrounded by double-quotes to contain an embedded double-quote, represented by a pair of consecutive double-quotes.
- 5 A CSV string may be terminated by 0x0A (line feed) or by 0x0D,0x0A (carriage return, line feed).

AUTHOR

Alan Citterman <alan@mfgrtl.com>

SEE ALSO

perl(1)

Reading for Next Class

Llama book: Chapter 19 (CGI Programming)

Homework Assignment

1. Write an elegant piece of Perl code. It might be some particularly elegant idiom that we have not discussed (and which you believe is not in general use in the Perl community). It might be a particular feature of Perl that you believe is so incredibly useful, that you have been waiting for just the right place to use it.

Here are the rules of the game:

- Hand in a single (one-sided) sheet of paper with your answer. Make sure it has your name(s).
- Your answer need not be a full program – a fragment of code is fine.
- Your code does not need to be run. You do *not* need to submit sample input and output.
- You may use ellipses (...) for sections of code which are not relevant (such as the body of a loop, if it is not relevant to your example).
- Your comments should explain why the code is elegant and/or interesting.

Hand in your single-page example at the beginning of next session.

2. Download the handout for Session 8. This should be available at <http://www.goss.com/perl> by Wednesday, July 12 at 9AM. Bring a printout to Session 8.¹

Notes: ¹Not to be handed in.

This handout is Copyright © 2000 Clint Goss, All Rights Reserved.