

Beginning Perl Programming (X52.9543.001)**An Implementation of Homework 5**

Here's **one** implementation of Homework #4. The **bold** items highlight interesting areas of the code:

webError.pl

```
#!/App/Perl/bin/perl -w
#
# webError.pl
#
# Perl script to produce a summary report of web site errors from an
# Apache error log file.
#
# (c) Copyright 2000 by Clint Goss, All Rights Reserved.

use strict 'vars';

# Pick up our options

use Getopt::Std;
use vars qw($opt_d $opt_D $opt_f $opt_h $opt_v);
if (! getopts("dDf:hv")) { &usage(); exit; }

sub usage {
    print <<END_OF_HELP;
usage: $0 [options]

Produce a summary report of web site errors from an Apache error log file.

Options:
-d          Turn on moderate debugging
-D          Turn on very verbose debugging
-f FILE    Specify the error file. Default is error_log
-h          Print this help message
-v          Turn on verbose reporting mode
END_OF_HELP
}

if ($opt_h) { &usage(); exit; }

# Parameters to this program

my ($debug) = $opt_D ? 2 : ($opt_d ? 1 : 0);
my ($fileName) = $opt_f ? $opt_f : "error_log";
my ($verbose) = $opt_v;

# Open the input file. Here's a sample line from this file (note that it has
# been broken over two comment lines for readability):
#
# [Mon Jun 5 15:54:33 2000] [error] [client 216.52.234.14] File does ...
# ... not exist: /usr/local/apache/htdocs/_themes/indust/indtextb.jpg

open (ERROR_FILE, "<$fileName") || die "Can't open $fileName: $!";
```

```

# Process the lines of the error file

my ($eFirstDate, $eLastDate);

my ($eLineCount) = 0;
my ($failMatchLineCount) = 0;

my (%typeCount);
my ($eFileLine);

my (%detailCrit, %detailError);

while ($eFileLine = <ERROR_FILE>) {

    $eLineCount++;

    # Extract the fields of this error record. The format is shown above.

    chomp $eFileLine;
    $eFileLine =~ /\[([^\]]+)\] \[([^\]]+)\] \[([^\]]+)\] (.*)/;
    my ($eDate, $eType, $eClient, $eDetail) = ($1, $2, $3, $4);

    if (! $eDetail) {
        $failMatchLineCount++;
        print "match failed on $eFileLine\n" if $debug;
        next;
    }

    if ($debug >= 2) {
        print "$eFileLine\n";
        print " eDate:  $eDate\n";
        print " eType:  $eType\n";
        print " eClient: $eClient\n";
        print " eDetail: $eDetail\n";
    }

    # Remember first and last date seen in the file.
    # ASSUMPTION: File ordered by date (reasonable, since it's a log file).

    $eFirstDate = $eDate unless $eFirstDate;
    $eLastDate = $eDate;

    # Record the count of each type of error and the count of
    # each particular detail of error.

    $typeCount{$eType}++;
    if ($eType eq "crit") { $detailCrit{$eDetail}++; }
    if ($eType eq "error") { $detailError{$eDetail}++; }

    # TBD: Record errors by the client's IP address.
    # TBD: Group all 'permission denied' errors for readability
}

```

```

# Clean up the double spaces in the dates.

$eFirstDate =~ s/ / /g;
$eLastDate =~ s/ / /g;

# The final report

print <<END_OF_HEADER;
Web site errors from $eFirstDate to $eLastDate

Web server error file: $eFileName
Errors reported: $eLineCount
END_OF_HEADER

if ($failMatchLineCount) {
    print "Warning: $failMatchLineCount lines in the file $eFileName\n";
    print "did not have a recognized format.\n";
}

print "\nTypes of errors reported [followed by count]:\n";
my ($eType);
foreach $eType (sort keys %typeCount) {
    printf "%s [%d]\n", $eType, $typeCount{$eType};
}

if ($verbose) {
    print "\nCritical errors, ordered by decreasing frequency:\n";
    my ($freq);
    foreach $freq (reverse sort values %detailCrit) {
        my ($eDetail);
        foreach $eDetail (keys %detailCrit) {
            my ($count) = $detailCrit{$eDetail};
            next if $count != $freq;
            printf "%s [%d]\n", $eDetail, $count;
        }
    }

    # TBD: Report non-critical errors
    # TBD: Report errors by client's IP address
}

exit;

```