**REPORT**

**On**

**DETAILED EXPLANATION**

**Of**

**SYSLOG PARSING SCIPTS USING PERL**

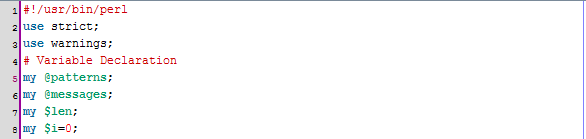
**SCRIPT-1**

**Purpose:**

**Parse the syslog messages file “/var/log/messages” and search for a keyword pattern given in a pattern file “keyword.txt”**

We will use the necessary PERL libraries and declare the necessary variables.

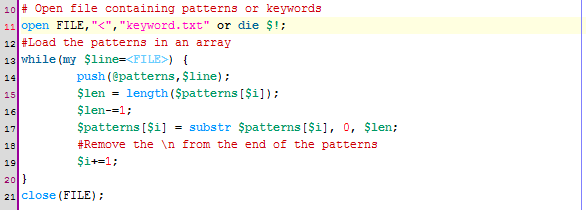
**Section1: Include required libraries and declare necessary variables:**



In the line 5 and 6 we declared two arrays to hold the patterns and lines from the “messages” log file respectively.

As a next step, the patterns (keywords) present inside “keyword.txt” are loaded into an array. It is assumed that each line contains a single pattern or a keyword that will be used to search in the logs file (“/var/log/message”) for a matching line.

**Section 2: Load all the keywords into an array:**

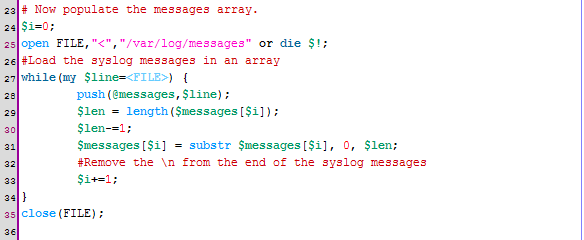


At the line number 11 in the above image, we read the keywords file ‘keyword.txt’ into a file handler ‘FILE’.

Next we use a while loop to read each line of the file and populate the array named “@patterns”. At line number 17 we use the substring function of PERL to exclude the last character “\n” which indicates a new line.

**Section3: Load all the lines from the log file into an array:**

To accomplish this we use another while loop, a file handler and an array to populate the array with lines inside the log file.



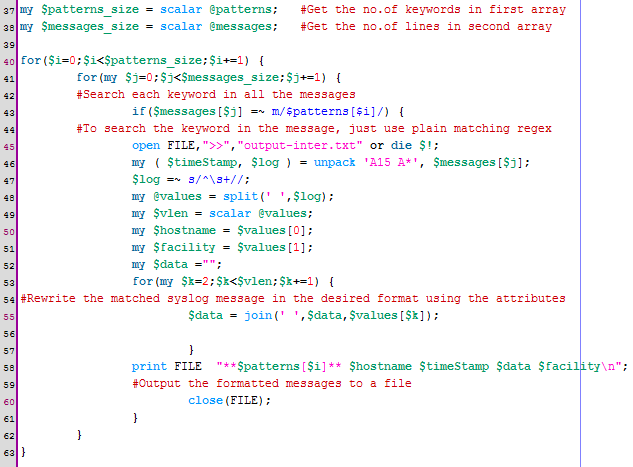
In the above section of code, we use another file handler to load the input file “/var/log/messages”.

Then we use another while loop to iterate through each line and populate it into the array called “@messages”. Again we delimit each line to strip the newline character at the end of each line. This is accomplished using lines 29-33 in the above section of code.

**Section 4 : Search for each keyword in the array containing message logs.**

Now that we have created and populated the two arrays containing keywords and log messages respectively, we are ready to start searching for each keyword (or pattern) from the first array and for a matching line in the second array.

We use below section of code to accomplish this:



In the above section of code at lines 27 and 28, we calculate the sizes of both arrays and store them in two variables.

The first for loop at line 40 above is used to iterate through each keyword in the keywords array. Then we use a nested for loop at line 41 to iterate through each message line in the second array containing messages.

At line 43 we search each line for a matching pattern containing the keyword at current index in the keywords array. We also set an “if” condition at the same line to get a Boolean result of either True or False.

If we get a matching line in the messages array, we want to add it to an output file called: “output-inter.txt”. Therefore we open another file handler at line number 45 pointing to our output file.

Here we need to understand the format of messages present in the log file so that the attributes could be parsed correctly. Below is the format used by syslog messages file:

<TIMESTAMP> <HOSTNAME > <REPORTING SERVICE/FACILITY> < DETAILED MESSAGE>

The timestamp contains date and time values in the format : MON DD HH:MM:SS

So in our script above we collect the first 15 characters at the beginning of each message into our timestamp variable. This is accomplished in the line number 46 of the code. As we can see that after the 15 characters containing the timestamp value, there is a white space which needs to be removed from the remaining part of the message. This is done in the line number 47 in above section of code.

Now we need to read the hostname and the reporting facility name into the two variables from the message line. To do this we delimit the message using space as a delimiter. It is done in the line number 48 in the above code snapshot.

After collecting the hostname as well as the reporting facility the remaining part in the message line is the detail but it is now delimited into multiple fields as it contained spaces. To overcome this , we rejoin all the remaining fields into a single variable ($data in our case).

To accomplish it, we iterate through each member of array called @values but start from array index 2. This is because we already picked the members at index 0 and 1 as hostname and facility/reporting service name. We combine all the members from index2 up to till the last index into a single variable called $data. This is done in line number 55.

Finally, we are ready to write all the variables back to our output file in the desired format. This is done in line number 58.

**SCRIPT 2**

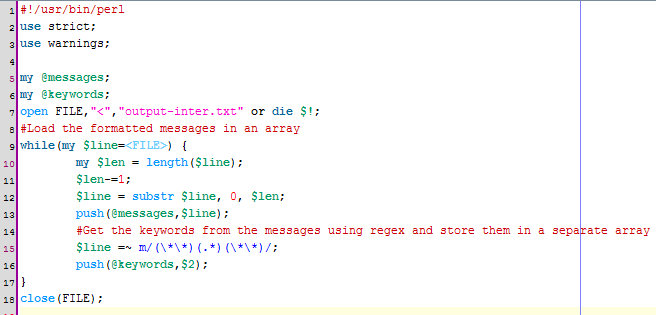
**Purpose :**

**Run a bubble sort on the search keywords and remove duplicate entries from the messages. Combine similar messages appearing in different date/time values.**

The output file produced by the Seeplogs.pl is used here as an input file. The lines are loaded into arrays and keywords are separated from the messages. Then the messages are sorted according to keywords.

**Section1: Load input file and populate the keywords array:**

In the below section of code, we use a file handler to open the file produced by the earlier script. We then use a while loop to iterate through each line and extract the keyword enclosed between \*\* characters. In line 16 we populate the array with the keyword.



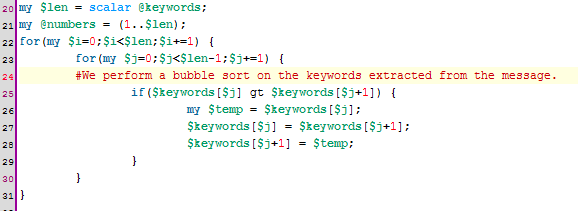
**Section 2 : Sort the lines bases on the keywords at the beginning of each line in the logs**

The overall objective of script2 is to filter out the logs based upon the keywords and display the output in a sorted order. To achieve this first we sort the keywords which we already loaded into an array. We use a bubble sort method here.

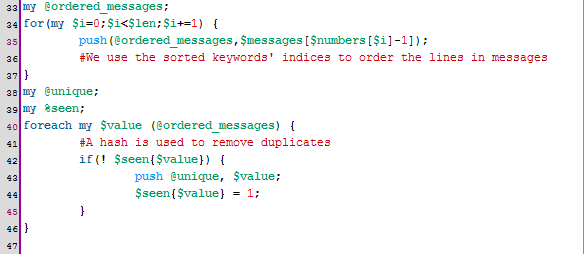
A temporary number array is used to store the index numbers of keywords which will help us in sorting the elements.

We create an array and name it @numbers that contains positive integer values from 1 upto total length of the keywords array. The we two loops (nested) to iterate through the array of keywords. The outer loop examines array elements from start till end while the inner loop goes up to the second last element of the @keywords array.

In the inner loop, we make a comparison of an element with its next neighbor and see if it is greater than its next neighbor.

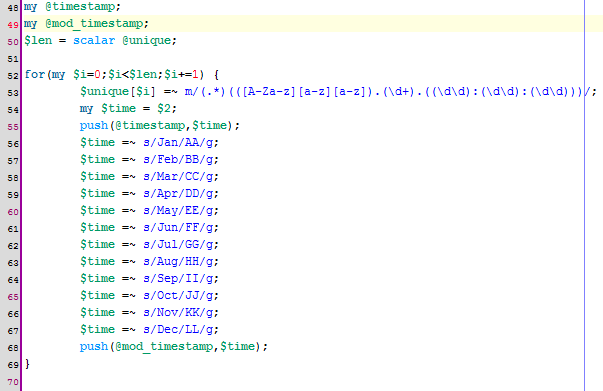


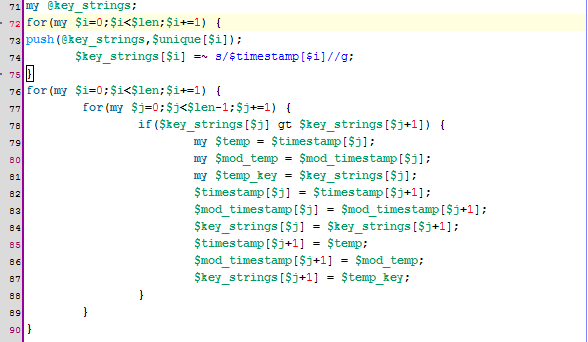
Next we push the ordered messages into another array called @ordered\_messages. And use a hash to delete the duplicate entries.



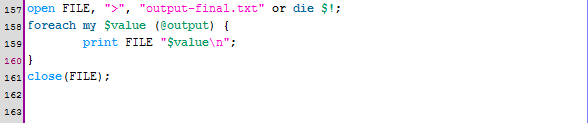
New process the unique messages and combine/group same messages occurring in different timestamp.

We show the number of merged lines after Hostname of the logging system (preceeded by a hash) and the date-time field converted to <start date> - <end-date>





And finally we write the output to the final output file.



**TECHNIQUES AND METHODS USED IN BOTH SCRIPTS**

**Script-1 & Script-2**

**Functions Used:**

**Push ()** and **Pop()** functions have been applied on arrays to add and remove members to arrays respectively. This can be found in line 13.

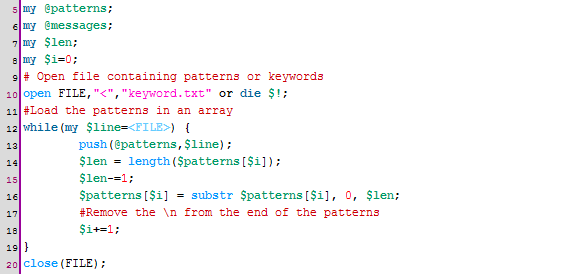
**Length()** function has been used to find the length of an array or to calculate the total number of elements in an array. (line14)

**Substr()** (substring) function has been applied to trim a larger string into smaller one or in this case we used it to remove the trailing newline characters. (line 16)

**Split()**function has been used to delimit fields. We used space as a delimiter character here. (line46 below)

**Upack** method is used here to get the first 15 characters from a line. (line 44 below)

**Join()** function has been applied to concatenate strings .(line 53 below)



**Loops:**

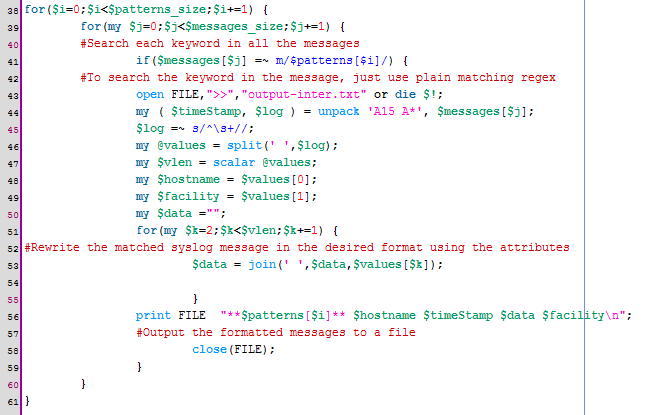
While loops have been used to iterate through an array or a file. (line 12)

**Nested Loops:**

Nested for loops have been used to compare the elements of two arrays. (lines 38 and 39 )

**File Handlers:**

We have used file handlers for opening files both in reading mode as well as writing mode. Arrays have been used to read the keywords or lines from the file so that these keywords or lines can be processed later on.

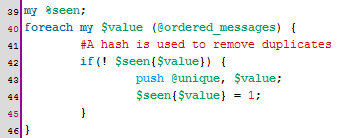


**Regular expressions:**

Regular expression based substitution has been used for search and replace keywords.(line 45)

**Hashes:**

We have used hashes to keep key/value pairs. We used this technique to remove duplications in messages log file. In below section we created a hash named ‘seen’ .



APPENDEX-A

# Script-1

#!/usr/bin/perl

use strict;

use warnings;

# Variable Declaration

my @patterns;

my @messages;

my $len;

my $i=0;

# Open file containing patterns or keywords

open FILE,"<","keyword.txt" or die $!;

#Load the patterns in an array

while(my $line=<FILE>) {

push(@patterns,$line);

$len = length($patterns[$i]);

$len-=1;

$patterns[$i] = substr $patterns[$i], 0, $len;

#Remove the \n from the end of the patterns

$i+=1;

}

close(FILE);

# Now populate the messages array.

$i=0;

open FILE,"<","/var/log/messages" or die $!;

#Load the syslog messages in an array

while(my $line=<FILE>) {

push(@messages,$line);

$len = length($messages[$i]);

$len-=1;

$messages[$i] = substr $messages[$i], 0, $len;

#Remove the \n from the end of the syslog messages

$i+=1;

}

close(FILE);

my $patterns\_size = scalar @patterns; #Get the no.of keywords in first array

my $messages\_size = scalar @messages; #Get the no.of lines in second array

for($i=0;$i<$patterns\_size;$i+=1) {

for(my $j=0;$j<$messages\_size;$j+=1) {

#Search each keyword in all the messages

if($messages[$j] =~ m/$patterns[$i]/) {

#To search the keyword in the message, just use plain matching regex

open FILE,">>","output-inter.txt" or die $!;

my ( $timeStamp, $log ) = unpack 'A15 A\*', $messages[$j];

$log =~ s/^\s+//;

my @values = split(' ',$log);

my $vlen = scalar @values;

my $hostname = $values[0];

my $facility = $values[1];

my $data ="";

for(my $k=2;$k<$vlen;$k+=1) {

#Rewrite the matched syslog message in the desired format using the attributes

$data = join(' ',$data,$values[$k]);

}

print FILE "\*\*$patterns[$i]\*\* $hostname $timeStamp $data $facility\n";

#Output the formatted messages to a file

close(FILE);

}

}

}

SCRIPT-2

#!/usr/bin/perl

use strict;

use warnings;

my @messages;

my @keywords;

open FILE,"<","output-inter.txt" or die $!;

#Load the formatted lines in an array

while(my $line=<FILE>) {

my $len = length($line);

$len-=1;

$line = substr $line, 0, $len;

push(@messages,$line);

#separate the keywords from the file and store them in a separate array

$line =~ m/(\\*\\*)(.\*)(\\*\\*)/;

push(@keywords,$2);

}

close(FILE);

my $len = scalar @keywords;

my @numbers = (1..$len);

for(my $i=0;$i<$len;$i+=1) {

for(my $j=0;$j<$len-1;$j+=1) {

#We perform a bubble sort on the keywords extracted from the message.

if($keywords[$j] gt $keywords[$j+1]) {

my $temp = $keywords[$j];

$keywords[$j] = $keywords[$j+1];

$keywords[$j+1] = $temp;

}

}

}

my @ordered\_messages;

for(my $i=0;$i<$len;$i+=1) {

push(@ordered\_messages,$messages[$numbers[$i]-1]);

#We use the sorted keywords' indices to order the lines in messages

}

my @unique;

my %seen;

foreach my $value (@ordered\_messages) {

#A hash is used to remove duplicates

if(! $seen{$value}) {

push @unique, $value;

$seen{$value} = 1;

}

}

my @timestamp;

my @mod\_timestamp;

$len = scalar @unique;

for(my $i=0;$i<$len;$i+=1) {

$unique[$i] =~ m/(.\*)(([A-Za-z][a-z][a-z]).(\d+).((\d\d):(\d\d):(\d\d)))/;

my $time = $2;

push(@timestamp,$time);

$time =~ s/Jan/AA/g;

$time =~ s/Feb/BB/g;

$time =~ s/Mar/CC/g;

$time =~ s/Apr/DD/g;

$time =~ s/May/EE/g;

$time =~ s/Jun/FF/g;

$time =~ s/Jul/GG/g;

$time =~ s/Aug/HH/g;

$time =~ s/Sep/II/g;

$time =~ s/Oct/JJ/g;

$time =~ s/Nov/KK/g;

$time =~ s/Dec/LL/g;

push(@mod\_timestamp,$time);

}

my @key\_strings;

for(my $i=0;$i<$len;$i+=1) {

push(@key\_strings,$unique[$i]);

$key\_strings[$i] =~ s/$timestamp[$i]//g;

}

for(my $i=0;$i<$len;$i+=1) {

for(my $j=0;$j<$len-1;$j+=1) {

if($key\_strings[$j] gt $key\_strings[$j+1]) {

my $temp = $timestamp[$j];

my $mod\_temp = $mod\_timestamp[$j];

my $temp\_key = $key\_strings[$j];

$timestamp[$j] = $timestamp[$j+1];

$mod\_timestamp[$j] = $mod\_timestamp[$j+1];

$key\_strings[$j] = $key\_strings[$j+1];

$timestamp[$j+1] = $temp;

$mod\_timestamp[$j+1] = $mod\_temp;

$key\_strings[$j+1] = $temp\_key;

}

}

}

my $previous = "";

my $count = 0;

my $i=0;

my @times;

my @output;

foreach my $value (@key\_strings) {

if($seen{$value}) {

push(@times,$mod\_timestamp[$i]);

$count+=1;

} else {

$seen{$value} = 1;

$len = length($previous);

if($len>1) {

my @tokens = split(' ',$previous);

my $words = scalar @tokens;

my $data = "";

$data = join(' ',$data,$tokens[0],$tokens[1]);

my $count\_string = "#".$count;

my @sorted\_time = sort @times;

my $start\_time = $sorted\_time[0];

my $end\_time = $sorted\_time[(scalar @sorted\_time)-1];

$start\_time =~ s/AA/Jan/g;

$start\_time =~ s/BB/Feb/g;

$start\_time =~ s/CC/Mar/g;

$start\_time =~ s/DD/Apr/g;

$start\_time =~ s/EE/May/g;

$start\_time =~ s/FF/Jun/g;

$start\_time =~ s/GG/Jul/g;

$start\_time =~ s/HH/Aug/g;

$start\_time =~ s/II/Sep/g;

$start\_time =~ s/JJ/Oct/g;

$start\_time =~ s/KK/Nov/g;

$start\_time =~ s/LL/Dec/g;

$end\_time =~ s/AA/Jan/g;

$end\_time =~ s/BB/Feb/g;

$end\_time =~ s/CC/Mar/g;

$end\_time =~ s/DD/Apr/g;

$end\_time =~ s/EE/May/g;

$end\_time =~ s/FF/Jun/g;

$end\_time =~ s/GG/Jul/g;

$end\_time =~ s/HH/Aug/g;

$end\_time =~ s/II/Sep/g;

$end\_time =~ s/JJ/Oct/g;

$end\_time =~ s/KK/Nov/g;

$end\_time =~ s/LL/Dec/g;

my $tstamp = join('-',$start\_time,$end\_time);

$data = join(' ',$data, $count\_string, $tstamp);

for(my $i=2;$i<$words;$i+=1) {

$data = join(' ',$data,$tokens[$i]);

}

push(@output,$data);

for(my $i=0;$i<$count;$i+=1) {

pop(@times);

}

$previous = $value;

push(@times,$mod\_timestamp[$i]);

$count = 1;

} else {

$previous = $value;

push(@times,$mod\_timestamp[$i]);

$count+=1;

}

}

$i+=1;

}

open FILE, ">", "output-final.txt" or die $!;

foreach my $value (@output) {

print FILE "$value\n";

}

close(FILE);