faqgrep: A Jumpstart Perl Script

A. Johnson

November 14, 1999

Abstract

This little article is intended as a 'jumpstart' script into learning Perl. The goal is not to try to teach Perl programming in one article or with one script, but to try to give an entry point into learning Perl with a script that should also prove useful as you continue your learning process. This document is actually a 'Literate Program' written using the `noweb` system — for those who aren't familiar with reading code in this form, an appendix provides some explanation and pointers to further information on `noweb` and literate programming.

1 Introduction

Perl is a very powerful programming language which excels at (but is not limited to) text processing. The Perl language is not a difficult language, though it can seem cryptic at first blush due to all the funky punctuation which appears in the language. Like any language, it merely takes time and practice to become familiar with the syntax and meanings and the ways of saying different things (or saying the same thing differently).

The intent of this article is not to teach Perl, but rather to introduce a single, well documented script `faqgrep`. The bonus is that unlike many of the example scripts in books and tutorials, this is a script that you can (and should) continue to use to help you use perl more effectively. Programming is about problem solving. Very often one begins to tackle a problem by breaking it down into smaller problems and addressing those. However, when learning a new language, some of the smaller problems are often difficult because you do not yet know how to express the solution in the context of the language you are learning. This is when it is time to turn to the FAQ's.

Do not assume that the FAQ's only address simple or "little" questions and that your question will not be found there—many FAQ's do have simple answers, but they are no less valuable for that. On the other hand, there are also many "real programming issues" addressed in Perl's FAQ's, so not matter how easy or difficult your particular problem seems to be, you'll often have good luck finding something of use in the FAQ's.

The script presented here is designed to facilitate searching through the nine sections of Perl's FAQ's for questions (and answers) relating to a keyword or phrase you supply on the command-line. The script is not long (less than 40 lines of actual code) but it does demonstrate some common structures and commands used in processing textfiles.

2 `faqgrep`: What is it? an informal specification

The Perl FAQ's are broken into 9 sections, each covering some related set of topics, and are included with any proper perl distribution. They come in the form of POD source files which can be read using the `perldoc` tool, or converted to other formats such as manpages, html, latex, or plain text using various translators (also included with perl).

Let us say that we have some kind of question about sorting. We'd rather not browse through each FAQ looking for sort related information, and perhaps we don't want to find every single instance of the word `sort` for it may occur in contexts unrelated to sorting at all — "This sort of thing..."— or may be used in contexts which do not directly answer questions about sorting.

In the POD source files, each question is a second level heading which appears like:

```
=head2 How do I ...  in Perl?
```

so we know, when we are looking for questions we need only look at lines beginning with `=head2`. Thus, we could always use the `grep` tool on the command-line like:

```
grep '=head2.*sort' /path/to/perl_pods/*faq*.pod
```

and we would find out that `perlfaq4.pod` contained 3 questions containing the word "sort", and we could then proceed to read perlfaq4. The `faqgrep` script presented here simply automates the above task, and provides a little more functionality. With it we can do the equivalent search from the command-line:

```
faqgrep sort
```

But, we can also supply a command-line option to tell `faqgrep` to print out the answer to all of the found questions, or we can adjust our search pattern to focus on the question we really want. For example, let's assume we have a question on how to get a line randomly from a file:

```
[danger:ajohnson:~/devel/noweb/faqgrep]$ faqgrep random
perlfaq4.pod:=head2 Why aren't my random numbers random?
perlfaq4.pod:=head2 How do I shuffle an array randomly?
perlfaq4.pod:=head2 How do I select a random element from an array?
perlfaq5.pod:=head2 How do I randomly update a binary file?
perlfaq5.pod:=head2 How do I select a random line from a file?
```

Now, we see that the last question returned is the one we want so we add another keyword that only occurs in that question and use the `-f` option to `faqgrep` to tell it to extract the answer:

```
[danger:ajohnson:~/devel/noweb/faqgrep]$ faqgrep -f random line
perlfaq5.pod:=head2 How do I select a random line from a file?

Here's an algorithm from the Camel Book:

    srand;
    rand($.) < 1 && ($line = $_) while <>;
'blah'
This has a significant advantage in space over reading the whole
file in.  A simple proof by induction is available upon
request if you doubt its correctness.
```

This demonstrates the default behavior when using more than one keyword—to search for entries containing **all** of the keywords. Alternately, one might wish to find all entries matching **any** of the keywords—this is done with the `-or` option and can result in quite a lot of output if one is not careful.

3 faqgrep: The Script

To begin with, we will first outline the overall structure of the script:

3a ⟨*faqgrep* 3a⟩≡
```
#!/usr/bin/perl -w
use strict;
⟨initialize_variables 3b⟩
⟨read_in_perlfaq_filenames 4b⟩
⟨search_and_print 4c⟩
⟨faqgrep.pod 5b⟩
```

All we have done thus far is subdivide our program into a few smaller chunks which we can address one at a time, subdividing further if need be. The first thing we need to do is initialize some varriables. In particular, we need a variable to store the location of the perlfaq POD files, two variables for our options, and one to hold our pattern.

3b ⟨*initialize_variables* 3b⟩≡ (3a)
```
# set $faqdir to point to your installed pod files:
my $faqdir = '/usr/local/lib/perl5/5.00503/pod';
my($opt_f,$opt_or,$pattern);
⟨get_options 3c⟩
⟨create_keyword_pattern 4a⟩
```

Defines:
`$faqdir`, *used in chunk 4.*
`$opt_f`, *used in chunks 3c and 5a.*
`$opt_or`, *used in chunks 3c and 4a.*
`$pattern`, *used in chunks 4 and 5a.*

We use a simple method to extract our command line options and `die` with a usage statement if we get an unrecognized option:

3c ⟨*get_options* 3c⟩≡ (3b)
```
while($ARGV[0]=~/^-/){
    $_=$ARGV[0];
    if (/^-or$/){$opt_or=1;shift @ARGV;next}
    if (/^-f$/){$opt_f=1;shift @ARGV;next}
die<<HERE;
illegal option: $_
usage: faqgrep [-f] [-or] [keywords...]
HERE
}
```

Uses `$opt_f` *3b and* `$opt_or` *3b.*

Now that our options are removed from the command line, we can do a quick check for remaining keywords (or why bother running). To create our `$pattern` in the case of the `-or` option we merely join all the keywords together with the `|` regex alternation operator preceeded by a wildcard pattern. This is a somewhat slow approach, but it serves for the present task. For the default ANDed pattern, use a series of look-aheads, one for each keyword, which all must pass for the pattern to succeed.

4a ⟨*create_keyword_pattern* 4a⟩≡ (3b)

```
die "no keywords specified\n" unless @ARGV;
if($opt_or){
    $pattern = '.*(?:' . join('|',@ARGV) . ')';
}else{
    $pattern=join('',map{"(?=.*$_)"}@ARGV);
}

-----
```

Uses `$opt_or` *3b and* `$pattern` *3b.*

There are now 9 sections to the perlfaqs, named 'perlfaq1.pod' .. 'perlfaq9.pod'— We use `opendir` and `readdir` to get a list of the files in the `$faqdir`, and extract only those with the substring 'faq' in their names using the `grep` function:

4b ⟨*read_in_perlfaq_filenames* 4b⟩≡ (3a)

```
opendir(FAQDIR,$faqdir)|| die "can't open $faqdir $!";
my @faqs = grep /faq/,readdir FAQDIR;
closedir FAQDIR;

-----
```

Defines:
`@faqs`, *used in chunk 4c.*
Uses `$faqdir` *3b.*

Now we know what we are looking for, and what files to look in. We search through each faq file for lines beginning with `=head2` that are followed by our `$pattern`, and we print out the filename, and the matching line. If we found the `-f` option on the command line, we also print the answer:

4c ⟨*search_and_print* 4c⟩≡ (3a)

```
foreach my $faq (@faqs) {
    open(FAQ,"$faqdir/$faq")||die "can't $!";
    while (<FAQ>) {
        if (s/^=head2($pattern)/$1/io) {
            print "$faq:$_" ;
            ⟨print_answer_if_-f_option 5a⟩
        }
    }
    close FAQ;
}

-----
```

Uses `$faqdir` *3b,* `@faqs` *4b, and* `$pattern` *3b.*

To print the answer, we just keep printing lines until we hit another `=head2` line that does not contain our pattern:

5a ⟨*print_answer_if_-f_option* 5a⟩≡ (4c)
```
  if ($opt_f) {
      while(<FAQ>){
          last if m/^=head(?!$pattern)/io;
          print;
      }
  }
```
Uses `$opt_f` *3b and* `$pattern` *3b.*

POD: faqgrep.pod

NAME

faqgrep — perl script to search perlfaqs

SYNOPSIS

```
    faqgrep [-f] [-or] [keywords...]
    faqgrep sort hash
    faqgrep -or sort hash
    faqgrep -f sort array
```

DESCRIPTION

This script takes keywords as arguments and searches through the perlfaqs printing on STDOUT the questions (and optionally the answers) which contain the keywords.

-or
: By default, keywords are 'ANDed' together to find questions which contain ALL of the keywords. Using the **-or** option 'ORs' together the keywords to find questions which contain ANY of the keywords.

-f
: By default only the matching questions are printed. Using this option causes the full entry (answer) of all matching questions to be printed. This can cause a lot of output unless the search was restricted. It is recommended that you first search for just one or two keywords and see the resulting matches and then repeat the search using **-f** and additional keywords to restrict output to just those entries you wish to see.

AUTHOR

Andrew L. Johnson <andrew-johnson@home.com>

COPYRIGHT

Copyright 1997−1998 Andrew L Johnson. This is free software and you may redistribute it and/or modify it under the same terms as Perl itself.

5b ⟨*faqgrep.pod* 5b⟩≡ (3a)
```
  This pod chunk has been converted to latex,
  see above.
```

Identifier Index

Code Chunk Index

4 Improvements

There are many ways this code could be improved upon—so go ahead and hack the heck out of it... we all know it won't be complete until it reads email :-).