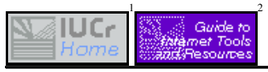

Perl for Crystallographers

Introduction to Perl

Perl, the **P***ractical* **E***xtraction and* **R***eport* **L***anguage* was created by Larry Wall in the early 1980's in order to provide an easy way to solve many common tasks under Unix that had previously required Unix shells, *awk* and *sed* scripting combinations to solve. As such **Perl** is an interpreted language optimized for easy manipulation of files, text, and processes and combines the power of *C* with the text manipulative capabilities of *awk* and *sed*, and without many of the limitations of any of these.

Because of its exceptionally powerful text, string and data manipulation capabilities, **Perl** is ideally suited to solving many programming problems typical in the World Wide Web's CGI (Common Gateway Interface). For example, if you encounter a form on the WWW, and fill it in, the data you send back to the server is most often dealt with by a particular **Perl** program. The 4th GVU WWW user survey[3] found that most people (46.7%) used **Perl** for CGI programming, making it by far the most commonly used CGI language[4] (*C* was second at 12.5%).

With its roots embedded in *C*, *awk*, *sed* and *sh*, **Perl** is capable of much that these languages are. It has a structure reminiscent of *C* with constructs like **if**, **for** and **while**, complemented with the powerful regular expressions of *sed*, *awk* and *lex*. However, unlike these UNIX utilities, **Perl** does not arbirarily limit the size of your data. **Perl** can read an entire file into a single string, allowing very powerful manipulation of that data. Also, recursion has no depth restrictions.

Although, at first glance, **Perl** appears to be a language designed for text manipulation, and therefore not suitable to *serious* tasks, it should be pointed out that not only can it also handle binary data with similar ease, but it also has features for networking and security.

It is, then, no surprise that more than half the common CGI programs witten for the WWW are written in **Perl**.

To find out more about **Perl** in general, have a look at the University of Florida's **Perl** Archive at: http://www.cis.ufl.edu/perl[5]

Perl for Crystallographers

Although **Perl** has clearly been accepted as the language of choice for WWW/CGI programming, its primary design as a data manipulation language makes it an excellent programming platform for general crystallographic data control. We now present some of the most important applications.

Format conversion

Have you ever had structure factor tables in a format that was not suitable to input into your refinement package? No one wants to use a text editor to modify the format of several thousand lines of data. Few people would take the time to write a *C* program to do the required formatting. But with **Perl** the program could be as short as one line.

The program:

```
while(<>){
```

[1] http://www.iucr.ac.uk/iucr-top/welcome.html
[2] int.index.html
[3] http://www.cc.gatech.edu/gvu/user_surveys/survey-10-1995
[4] http://www.cc.gatech.edu/gvu/user_surveys/survey-10-1995/graphs/info/languages.html
[5] http://www.cis.ufl.edu/perl/

```
  s/^\s+//;s/\n//;($h,$k,$l,$f,$fc,$sf)=split(/\s/,$_);
  printf "%4d%4d%8.2f%8.2f\n",$h,$k,$l,$f/100.0,$sf/10.0;
}
```

would convert any number of data with an *h*, *k*, *l*, 100*F_o, 100*F_c, 10*sigF_o format like this:

```
5,4,2,1534,1486,15
5,4,3,134,139,11
```

into shelx HKLF3 formatted data like this:

```
   5   4   2   15.34    1.50
   5   4   3    1.34    1.10
```

This is a particularly simple example, of course, and could just as easily be done with *awk*. Very often, however, the structure factor code is in *condensed*[6] format separated across multiple pages with headings and other complicating factors. This is where **Perl**'s formatting power[7] becomes more obvious.

Data Extraction and Correlation

Programs can easily be written to search through files for key information and reformat it into suitable format. A good example of this would be a program to search **CAD4 raw data** files for reflections fitting specified *chi* ranges and intensity values, for the purposes of finding reflections geometrically suited to *psi–scans* for empirical absorption corrections.

Another good example would be processing **Cambridge Database** output files to search for correlations not available as part of the CSD system. For example you could have made a seach for all molecules with a certain fragment. The **Perl** program could be used to then calculate statistics on which authors published the most relevant papers, and in which years. Access to other databases, like the **World Directory of Crystallographers** would allow you to automatically email the most important authors for futher information where appropriate.

Program Control

Very often the same program needs to be run many times under similar circumstances. **Perl** can allow you to automate many complex but repetitive tasks simple and easy to manage. I was able to refine 34 complete crystal structures simultaneously, and make common changes to all 34 input data files with single commands. Of course, the final results were trivial to correlate using **Perl** as well, allowing me to view the final refinement parameters for all 34 structures in a well formated table. **Perl** also spawned the appropriate graphics programs to plot the correlated results in an easy to view manner.

This saved, not hours, but months of hard and tedious work.

For a simple example of this kind of facility, consider the following **Perl** code:

```
# Read four data files
@file1=<file1.dat>;@file2=<file2.dat>;@file3=<file3.dat>;@file4=<file4.dat>;

# find correct lines with "R-Factor ="
@Rl1=grep(/R-Factor =/,@file1);
@Rl2=grep(/R-Factor =/,@file2);
@Rl3=grep(/R-Factor =/,@file3);
@Rl4=grep(/R-Factor =/,@file4);

# open output file
open(FILE,">output.file");

# find actual R-Factors and save table of data in "output.file"
for($i=0;$i<scalar(@Rl1);$i++){
        @Rl1[$i]=~/R-Factor = +([0-9\.]+)/;  $R1=$1;
        @Rl2[$i]=~/R-Factor = +([0-9\.]+)/;  $R2=$1;
        @Rl3[$i]=~/R-Factor = +([0-9\.]+)/;  $R3=$1;
        @Rl4[$i]=~/R-Factor = +([0-9\.]+)/;  $R4=$1;
        print FILE "$i  $R1  $R2  $R3  $R4\n";
}
```

[6] int.perl.hkl.code.html#table
[7] int.perl.hkl.code.html

```
close(FILE);

# run graphics program
open(PLOT,"|gnuplot");
print PLOT "plot \"output.file\" using 1:2 with linespoints, \"output.file\" using 1:3 with linespoints, \"output.file\" using 1:4 with linespoints, \"output.file\" using 1:5 with linespoints\npause -1\nquit\n";
```

This code reads lines from three files, finds the lines with the R–factors in them and saves a table of all results. It then calls the graphics program 'gnuplot' to plot a graph of the results for all four structures. Without comment lines, this program is only 16 lines long and took less than ten minutes to write, and provided on the screen a concise visual presentation of the sets of R–Factor resutls from four structures. That is real time efficiency.

Perl for Crystallographers on the WWW

However, it still remains that **Perl** is certainly most used by crystallographers on the WWW. The most common use involves registration forms[8], but many facilities that involve information exchange also use perl. The **World Directory of Crystallographers** can be accessed through a Search Form[9] that makes use of a perl CGI program.

Registration Forms

Recently it has become quite popular to register for conferences and other facilities through the WWW. This can either be a very simple automated email system, where the information entered by the person registering is written into an email and emailed to the conference organiser, or it could be a far more complex system, where the information is entered into a database that can then be administered by the conference organiser through the databases native client, or through the WWW client. In all cases, the language of choice for interfacing the WWW form to either a mail program or a database, is **Perl**.

A simple program to forward registration details to the conference organiser would look like this (assuming a POST form method):

```
# read registration information
read(STDIN,$information,$ENV{'CONTENT_LENGTH'});

# split up information
@lines=split(/&/,$information);
foreach $pair (@pairs){
        ($key,$value)=split(/=/,$pair);
        $value =~ tr/+/ /;
        $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
        $in{$key} = $value;
}

# mail organiser all registration information
open(MAIL,"|sendmail -f $in{'email'} -F $in{'fullname'} $organizer");
print MAIL "Subject: $in{'fullname'} registration for conference\n\n";
foreach $key(%in){print MAIL "$key  :  $in{$key}\n";}
close (MAIL);

# reply to WWW client
print "Content-type: text/html\n\n";
print "<html><head><title>Successful Registration</title></head><body>\n";
print "<h1>Successful Registration</h1>\n";
print "Thank you, $in{'fullname'}, for registering for this conference.\n";
print "Your information has been forwarded to the organisers.\n";
```

This 18 line program does not contain any of the error checking that you would use in a real **Perl** CGI program, but does emphasize the power of the language for this task.

Searching the WDC–9

The IUCr provides the World Directory of Crystallographers[10] in both printed form and in electronic form on the internet. There are several ways of accessing the database electronically and all revolve around the fact that the database is in the *qi* database format, which the

[8] http://www.gh.wits.ac.za/indaba/reg–indaba.html
[9] http://occam.gh.wits.ac.za/iucr/docs/wdc9t.html
[10] int.wdc.gen.html

text–based phonebook client *ph* can search. This program returns a text answer, allowing other programs to easily format the answer in a customisable way. In the case of the WWW, the CGI program needs to add the HTML formatting to the answer so that is can be appropriately viewed. The following two lines from an example CGI program would find the persons email address from the text, and display it in HTML with an embedded mail link:

```
($email)=grep(/email:\s*/,@all_lines);
print "<p>Email: <a href="\mailto:$1\">$1</a>\n" if($email=~/email:\s*(.*)\s*$/);
```

Any other programming language would take many more lines of complex coding to perform this simple task.

However, it is also true that the simpler the task, the simpler the language required for it, and the more complex the task, the more complex the langauge. For some of the simple examples described here, *Bourne*, and other, shell scripts would also work, but only in conjuction with other *Unix* facilities like *awk* and *sed*. A powerful programming language like *C* could definitely do all the examples on it's own, but would be very, very difficult to program, and resulting programs would be several times as long as the **Perl** equivalent.

In conclusion, **Perl** is certainly the ideal language for many tasks that crystallographers encounter both in the lab (ie. on the computer) and on the *Crystallographic Web*.[11]

[Index][12] [FoFc example][13] [Registration Form][14] [WDC9 Form][15]– 17th Sept. 1996 – © B. Craig Taverner[16] **– Not to be copied or reproduced without permission –** Author's current manuscript[17]

[11] int.index.html
[12] int.index.html
[13] int.perl.hkl.code.html
[14] http://www.gh.wits.ac.za/indaba/reg–indaba.html
[15] http://occam.gh.wits.ac.za/iucr/docs/wdc9t.html
[16] http://www.gh.wits.ac.za/craig/
[17] http://www.gh.wits.ac.za/~craig/aiucr/int.perl.html