Around the Perl in 11 days

Day 8: System calls.
Using Perl to launch a new process

• A project may require you to run a particular program many times and then parse its output.
• Such a program could be an external program (BLAST, Stride) etc. or even another Perl program
• Perl makes it easy to launch a new program from within it and collect its output..........
• .....and there are different ways to do it
Using back ticks ``

- One way to launch a shell command is to put the command within back ticks or backquotes.
  
  e.g.: $who_result = `who`;

  “who” is a Unix command that gives a list of users logged on to the system. The result of the command is now the value of $who_result;

- The command within the backquotes is a *child process* to the original Perl program.

- The *parent process* will wait for the *child process* to be completed before it continues.
Environment variables

- A child process inherits its parents environment variables, current directory, standard filehandles etc.

- Environment variables are a set of hidden values that a program inherits from the shell. Some examples:
  - $HOME - points to the home directory of the user
  - $PATH - gives a list of paths that the shell will search when looking for an executable

- It is always best to give the full path to the executable within the backquotes.
  - `$who_result = `/usr/bin/who`;
  - You can find the full path to the executable by using the “where” command on the command line.
A simple example

- A Perl program that lists the Perl scripts in your directory.

``` perl
#!/usr/bin/Perl -w
use strict;
my @files;
@files = `ls -al */*.pl`;
push (@files, `ls -al */*.prl`);#
print @files;
```
The *system* function

- Another simple way to launch a new process is to use the *system* function.
  - e.g. `system("who");`
- When the command is finished, the system function returns the exit value of the command
  - "0" if the command was executed without any errors
  - "1" if the command was not executed
- The output for the command executed by the system function goes to the standard output (as inherited from the Perl program). In most cases this is the users display screen
More on the *system* function

• Since the command is a shell command, you can redirect output as you would in a Unix shell (using “>”)
  
  - e.g. system(“who > userids”);
  - The output of the system command is now stored in a file called “userids” in the directory the program is being run from.

• The system command can also take a list of arguments rather than a single argument.
  
  - system “grep 'Mus musculus' day5.dat”;
  - system “grep”, ”Mus musculus”, ”day5.dat”;
A simple example

• A simple Perl program to “grep” for a user defined string in a given file

#!/usr/bin/Perl -w

use strict;

my $file_for_searching = $ARGV[0];
my $search_string = $ARGV[1];

system "grep \'$search_string\' $file_for_searching > search_output";
Try it yourself

- sim4 is a similarity-based tool for aligning an expressed DNA sequence (EST, cDNA, mRNA) with a genomic sequence for the gene.
  - sim4 seqfile1 seqfile2 {[WXKCRDAPNB]=value}
- If seqfile2 is a database of sequences, the sequence in seqfile1 will be aligned with each of the sequences in seqfile2.
- Copy the file cdna.tbl from ~bio01/data to your directory. The file has cDNA data of about 60 genes.
- Using sim4, you need to find the location of the genes on a genomic sequence which is given in the file chromosome.raw.
Try it yourself

- Run `sim4` and compare the sequences in `cdna.tbl` to the file `chromosome.raw` which is also in `~bio01/data`.

```
/usr2/bioin/bio01/bin/sim4 <fasta file> ~bio01/data/chromosome.raw
```

- `sim4` compares one sequence at a time to the genomic sequence.

- You will first need to retrieve sequences one at a time from `cdna.tbl`, convert the sequence to a fasta format and write it to a temporary file. You can then run `sim4` to compare that one sequence to the genomic sequence.
Try it yourself

- Example output from sim4

seq1 = b0011.fa, 714 bp

seq2 = ~bio01/data/chromosome.raw ((no header)), 4639221 bp

(complement) Whether complement or not

1-714 (10643-11356) 100%

start stop

The final output of the program will be a table

<table>
<thead>
<tr>
<th>id</th>
<th>length_of_match</th>
<th>start</th>
<th>stop</th>
<th>complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>b0001</td>
<td>66</td>
<td>190</td>
<td>255</td>
<td>W</td>
</tr>
<tr>
<td>b0011</td>
<td>714</td>
<td>10643</td>
<td>11356</td>
<td>C</td>
</tr>
</tbody>
</table>

(C for complement; W otherwise)