Learning Perl Through Examples
Part 2
L1110@BUMC
9/22/2017
Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
Tutorial Resource

Before we start, please take a note - all the codes and supporting documents are accessible through:

- http://rcs.bu.edu/examples/perl/tutorials/
Sign In Sheet

We prepared sign-in sheet for each one to sign
We do this for internal management and quality control
So please SIGN IN if you haven’t done so
Evaluation

One last piece of information before we start:

- DON’T FORGET TO GO TO:
  - http://rcs.bu.edu/survey/tutorial_evaluation.html

Leave your feedback for this tutorial (both good and bad as long as it is honest are welcome. Thank you)
Today’s Topic

• Basics on creating your code
• About Today’s Example
• Learn Through Example 1 – fanconi_example_io.pl
• Learn Through Example 2 – fanconi_example_str_process.pl
• Learn Through Example 3 – fanconi_example_gene_anno.pl
• Extra Examples (if time permit)
Basics on creating your code

How to combine specs, tools, modules and knowledge.

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services

www.perl.org
Fall 2017
What is needed

Consider your code/software a ‘product’, what will it take to produce it?

• User Requirements (domain knowledge, that’s very important)
• Development Environment (Emacs/gedit/Eclipse/etc)
• Third Party Modules/Toolboxes (CPAN)
• Some workman’s craft (You/Programmer)
• Help systems (Help documentation/reference books/stackflow/etc)
• Language specification (Perldoc/reference guide)
User Requirements

Specify what software is expected to do
Can be formal or casual, but better keep records of.

Formal – User Requirement Documentation (URD)
Casual – email conversations, scratch paper memos, etc.

Types of Requirements
M – Mandatory
D – Desirable
O – Optional
E – Enhanceable

Serve as contract – keep project on track
Pitfall – often ignored

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
Development Environment

It is like your workshop where you go to work and make your product

How to pick your development tools (mainly editor or IDE)

- Convenient
- Sufficient enough
- Extensible/adaptive
- Personal preference
Development Environment

Some commonly used tools:

1) Editor Only:
   - emacs
   - vim
   - gedit

2) IDE (Integrated Development Environment)
   - Eclipse
   - Padre

You may go to http://perlide.org/poll200910/ for the poll result conducted by a Perl guru for Perl Editors
CPAN – Where Third Party Modules Resides

- Perl is a community built software system, enriched by third party contributors. All efforts go to build CPAN open source archive network for Perl.

- Perl’s richness and power comes from CPAN and the 3rd party modules and toolkits covering various domains, for example, Finance, BioPerl, Catalyst, DBI, and many others.

- CPAN official site: www.cpan.org

- Two search engine interfaces:
  - search.cpan.org (old, traditional)
  - metacpan.org (new, modern, provides rich APIs for automation)
Help systems

One significant criteria for a good programming language is its documentation and help system – In this sense, Perl is quite good

Its own:

• Language Specification itself well written
• Organized well (divided by categories)
• Presented well (perldoc utility/man, Internet available)

Online Resource:

• Rich online help, tutorials, and e-books (many for free)
Language specification

Also called ‘Reference Guide’

Perldoc Official Site: http://perldoc.perl.org

Divided to eight subcategories:

1. Language
2. Functions
3. Operators
4. Special variables
5. Pragmas
6. Utilities
7. Internals
8. Platform Specific
Workman’s Crafts

Hard Part
Takes time to build, but takes no time to start (practice is the best way to learn)
Skills Needed Include:
• Familiarity to language elements
• Software Engineering Methodology
• Algorithm Design
• Code Implementation
• Debugging
• Domain knowledge

Metaphor: How do we acquire skills on natural language
Before We Start ...
Connecting to SCC

• Option 1: You are able to keep everything you generate
  Use your Shared Computing Cluster account if you have one.

• Option 2: all that you do in the tutorial may be wiped out after
  tutorial ends unless you move the contents to somewhere belong to
  you.
  Tutorial accounts if you need one (will be provided in class)
  Username : TBD
  Password : TBD
Download source code

Follow these steps to download the code:

```
ssh user@sccN.bu.edu ('user' is an account on SCC, 'N' can be 1-4)
mkdir perlThruEx
.cd perlThruEx
wget http://scv.bu.edu/examples/perl/tutorials/src/perlThruExamples.zip
```
Today’s Example Overview
Example Preview – Fanconi Gene Introduction

- Fanconi genes refer to the genes that have been identified as closely related to a genetic disease called ‘Fanconi Amaemia’ (FA).
- 17 genes are identified so far, and 15 of them named as ‘FANC[A-S]’, 2 others have totally non-revealing names, ‘RAD51C’ and ‘XPF’.
- For this example, we will only take the 15 genes that start with ‘FANC’ as the input gene list.
Example Preview – Content Coverage

This tutorial will use fanconi genes example to go through three main functional strengths of Perl – File IOs, string match and process, and last, power in using 3rd party modules, in this case, we use BioPerl’s gene annotation module, GenBank.

• Example 1: File IOs
• Example 2: String Processing
• Example 3: Gene Annotation
Example Preview – Code Organization

General Setting:

- Input directory – the place to put all input files
  - ./code/session2/data_in
- Output directory – the place where end result is put
  - ./code/session2/data_out
- Script directory – the place where Perl scripts reside
  - ./code/session2/scripts
Example 1 Preview

**Script:** fanconi_example_io.pl

**Purpose:** build up the standard File IO concepts.
- contains 6 subroutines, each demonstrates a slightly different way Perl handles input and output

**Command:**

```
perl fanconi_example_io.pl --example n
```

Note: fanconi_example_io_fancy.pl is a bit fancier version, which adds support of command line arguments for the flexibility
Example 2 Preview

**Script:** fanconi_example_str_process.pl

**Purpose:** Demonstrate regular expressions in Perl.

Contains 2 subroutines:
- input file format conversion, from csv to tab format
- gene selection using specified criteria, from all 15 genes, only pick first 5 with FANC[A-D].

**Command:** perl fanconi_example_str_process.pl --example n
Example 3 Preview

**Script:** fanconi_example_gene_anno.pl

**Purpose:** Demonstrate the comprehensive coding and debugging skills.
- Use BioPerl module (Bio::DB::EntrezGene) for gene annotation from entrez gene id.
- Go through code in detail with debugger

**Command:**

fanconi_example_gene_anno.pl

This will consume the most of the tutorial time.
Learn Through Example - File IO

fanconi_example_io.pl
File IO basics - Filehandle

A structure used to associate a physical file with name

Three standard file handles:

- **STDIN** – Standard input, usually set to be keyboard
- **STDOUT** – Standard output, usually set to be screen, using device id ‘1’
- **STDERR** – Standard error, to display error info (usually set to be same screen), using device id ‘2’

One special file handle - `/dev/null`, logical file handle to absorb all unwanted output, like black hole, no return once get in. For example:

```
> /dev/null 2>&1
```

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
File IO basics – File IO functions

• These are the actual functions one may use to manipulate files

• Basic File IO functions:

  • open
  • close
  • opendir
  • closedir
  • read
  • print
  • rename
  • unlink
  • tell
  • seek

Form full list:
http://perldoc.perl.org/index-functions-by-cat.html#Input-and-output-functions

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services

www.perl.org

Fall 2017
File IO basics - Operators

Operators are actually functions of special type, which are predefined by language to accomplish specific operations upon operand, usually independent of operand(s) it involves.

<> : File handle operator, used to read file using handle; can be single line or multiple lines, depending on context, for example:
   $single_line = <STDIN>; # read single line from screen
   @multi_lines = <DATA>; # read whole data file

- : File test operator, used to test various attributes of the file
   -e check file existence
   -s check file size
   -d check if file is actually a directory
   -z check if file size is zero
File IO basics – Special variables

There are many File IO related special variables in Perl.

For example:

- `$/` Input file line separator (delimiter)
- `$\` Output file line separator (delimiter)
- `$.` Line number
- `$|` No buffer flush (if set to true, flush right away)
- `$$` Error information
Common Scenarios involving file/data processing

1. Conversion between file formats.
   For example, convert Excel spreadsheet to plain text (.csv or .tab) for downstream processing (pre-processing).

2. Data filtering/cleaning/verification
   For example, clean and preview/summary the input data (pre-processing/in-processing)

3. Apply business logic to the clean/filtered input data file.

4. Output is not restricted to files. Could be a table in database, or memory block to feed the downstream in an integrated pipeline setting (but beyond this tutorial).
Input file – fanconi_genes.csv

```
[yshen16@scc4 script]$ cat ../data_in/fanconi_genes.csv
entrez_gene_symbol,entrez_gene_id
FANCA,2175
FANCB,2187
FANCC,2176
FANCD1,675
FANCD2,2177
FANCE,2178
FANCF,2188
FANCG,2189
FANCI,55215
FANCJ,83990
FANCL,55120
FANCM,57697
FANCN,79728
FANCO,5809
FANCP,84464
[yshen16@scc4 script]$  
```
File IO - Example 1 code (use default)

```perl
# Example 1: read with all default;
# read from file, then output to standard output device
# (screen in this case) explicitly
sub ex1_reflection {
    my($in_dir, $in_file) = @_; 
    
    # print out head info to mark the example 1: 
    print "IN Example 1:\n";
    print "\n";

    open IN, "<$in_dir\$in_file"; #open input file handler
    while(<IN>) {
        # $_ is the perl special variable to represent the current default
        # input line, it can be omitted(i.e. 'print;' is enough); but it
        # is a better practice to write down explicite code
        print $_; # could be simplifed as 'print;', omitted $_
        
    }
    close IN;

    print "OUT Example 1.\n";
    print "\n";
} # end ex1_reflection
```
File IO – Example 1 output

[yshen16@sc14 script]$ perl fanconi_example_io.pl --example 1
IN Example 1:
entrez_gene_symbol,entrez_gene_id
FANCA,2175
FANCB,2187
FANCC,2176
FANCD1,675
FANCD2,2177
FANCF,2178
FANCI,55215
FANCI,83990
FANCL,55120
FANCM,57697
FANCN,79728
FANC0,5889
FANCP,84464
OUT Example 1.
[yshen16@sc14 script]$
File IO – Example 2 code (explicit)

```perl
# Example 2: STDOUT concepts;
#
sub ex2_reflection_stdout {
    my($in_dir, $in_file) = @_;

    # print out head info to mark the example 2:
    print STDOUT "IN Example 2:
    print STDOUT "\n";

    open IN, "<$in_dir$in_file";  # open input file handler
    while(<IN>) {
        # explicitly specify STDOUT as the output device
        print STDOUT $_;
    }
    close IN;

    print STDOUT "OUT Example 2.";
    print STDOUT "\n";
} # end ex2_reflection_stdout
```
File IO – Example 2 output

```
[yshen16@scc4 script]$ perl fanconi_example_io.pl --example 1
IN Example 1:
entrez_gene_symbol,entrez_gene_id
FANCA,2175
FANCB,2187
FANCC,2176
FANCD1,675
FANCD2,2177
FANCE,2178
FANCF,2188
FANCG,2189
FANCI,55215
FANCI,83990
FANCL,55120
FANCM,57697
FANCN,79728
FANCO,5889
FANCP,64464
OUT Example 1.
[yshen16@scc4 script]$
```

Observation: compare Example 1 and Example 2, the output are the same.
File IO – Example 3 code (STDERR)

```perl
# Example 3: STDERR concepts;
# read from file, then output to standard error device
# (screen in this case, though) explicitly
sub ex3_reflection_stderr {
    my($in_dir, $in_file) = @_; 

    # print out head info to mark the example 3:
    print STDERR "IN Example 3:";
    print STDERR "\n";

    open IN, "<$in_dir$in_file"; # open input file handler
    while(<IN>) {
        print STDERR "; # explicitly specify STDERR as the output device
    }
    close IN;

    print STDERR "OUT Example 3:";
    print STDERR "\n";

} # end ex3_reflection_stderr
```
File IO – Example 3 output

<table>
<thead>
<tr>
<th>entrez_gene_symbol</th>
<th>entrez_gene_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>FANCA</td>
<td>2175</td>
</tr>
<tr>
<td>FANCB</td>
<td>2187</td>
</tr>
<tr>
<td>FANCC</td>
<td>2176</td>
</tr>
<tr>
<td>FANCD1</td>
<td>675</td>
</tr>
<tr>
<td>FANCD2</td>
<td>2177</td>
</tr>
<tr>
<td>FANCE</td>
<td>2178</td>
</tr>
<tr>
<td>FANCF</td>
<td>2188</td>
</tr>
<tr>
<td>FANCG</td>
<td>2189</td>
</tr>
<tr>
<td>FANCI</td>
<td>55215</td>
</tr>
<tr>
<td>FANCJ</td>
<td>83990</td>
</tr>
<tr>
<td>FANCL</td>
<td>55120</td>
</tr>
<tr>
<td>FANCN</td>
<td>57697</td>
</tr>
<tr>
<td>FANCN</td>
<td>79728</td>
</tr>
<tr>
<td>FANCO</td>
<td>5889</td>
</tr>
<tr>
<td>FANCP</td>
<td>84464</td>
</tr>
</tbody>
</table>

**Observation:** Example 3 seems to output the same result as Example 1 and Example 2.

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
File IO – Example 2 vs Example 3 output

Example 3 actually is not as same as example 1 and;
Example 1 and 2 -> STDOUT
Example 3 -> STDERR
File IO – Example 4 code (die)

```perl
# Example 4: die - display error message then quit the program
sub ex4_reflection_die {
    my($in_dir, $in_file) = @_;

    # print out header info to mark the example 4:
    print "IN Example 4:"
    print "\n";
    # open input file handler, deliberately change the input file name
    # by adding 'l' after the real file name.
    open IN, "<$in_dir\$in_file" . "l"
    or die "Sorry, can't open file " . $in_file . "l, $!";
    while(<IN>) {
        print $_
    }
    close IN;
    print "OUT Example 4:"
    print "\n";
} # end ex4_reflection_die
```
File IO – Example 4 output

Observation 1: using ‘die’ is a good practice when open file;
Observation 2: using ‘$!’ special variable shows the system error message;
File IO – Example 5 code (STDIN)

```perl
#*******************************************************************************
# Example 5: redirection and STDIN
#*******************************************************************************
sub ex5_reflection_stdin {
    # my($in_dir, $in_file) = @_;

    # print out head info to mark the example 5:
    print "IN Example 5:";
    print "\n";

    # get input from STDIN (keyboard), type ctrl-D to exit
    while(<>){
        print $_;
    }

    print "OUT Example 5.";
    print "\n";
}
# end ex5_reflection_stdin
```
File IO – Example 5 output

```
[yshen16@scc4 script]$ perl fanconi_example_io.pl --example 5
IN Example 5:
I am in Example 5,
I am in Example 5,
You copy me so faithfully
You copy me so faithfully
I am bored now
I am bored now
thank you.Bye-bye
thank you.Bye-bye
OUT Example 5.
[yshen16@scc4 script]$  
```
File IO – Example 5 output

```
[y Shen16@scc4 script]$ perl fanconi_example_io.pl --example 5
IN Example 5:
I am in Example 5,
I am in Example 5,
You copy me so faithfully
You copy me so faithfully
I am bored now
I am bored now
thank you. Bye-bye
thank you. Bye-bye
OUT Example 5.
[y Shen16@scc4 script]$
```
File IO – Example 5 redirect

```
[yshen16@scc4 script]$ perl fanconi_example_io.pl --example 5 < ../data_in/fanconi_genes.csv
IN Example 5:
entrez_gene_symbol,entrez_gene_id
FANCA,2175
FANCB,2187
FANCC,2176
FANCD1,675
FANCD2,2177
FANCE,2178
FANCF,2188
FANCG,2189
FANCI,55215
FANCJ,83990
FANCL,55120
FANCM,57697
FANCN,79728
FANCQ,5809
FANCP,84464
OUT Example 5.
[yshen16@scc4 script]$ perl fanconi_example_io.pl --example 5 < ../data_in/fanconi_genes.csv > copy_fanconi_genes.csv
[yshen16@scc4 script]$ diff copy_fanconi_genes.csv ../data_in/fanconi_genes.csv
id6
< IN Example 5:
18d16
< OUT Example 5.
[yshen16@scc4 script]$ 
```
File IO – Example 5 redirect

What extra we learn from Example 5:

Redirection is a very powerful mechanism in Linux;
It can make code a lot more flexible;
It is built upon the concept of Linux/Unix fundamentals –
everything is a file
File IO – Example 6 code (output to file)

```perl
# Example 6: write output to file instead of STDOUT/STDERR
*****************************
sub ex6_reflection_out {
    my($in_dir, $in_file, $out_dir, $out_file) = @_;

    # print out head info to mark the example 6:
    print "IN Example 6: ";
    print "\n";

    # open input file handler to read the input
    open IN, "<$in_dir\$in_file"
        or die "Sorry, can't open file " . $in_file . ", $!";
    open OUT, ">$out_dir\$out_file";
    while(<IN>) {
        print OUT ";_
    }
    close IN;
    close OUT;

    # now try to check if the output file is exactly same as the input: (it is supposed to):
    system("echo diff $out_dir\$out_file $in_dir\$in_file");
    my $exit_status = system("diff $out_dir\$out_file $in_dir\$in_file");
    system("echo no output is good output, exit with $exit_status");

    print "OUT Example 6: ";
    print "\n";
} # end ex6_reflection_out
```
File IO – Example 6 output

[yshen16@scc4 script]$ perl fanconi_example_io.pl --example 6
IN Example 6:
diff ../data_out/fanconi_genes_dup.csv ../data_in/fanconi_genes.csv
no output is good output, exit with 0
OUT Example 6.
[yshen16@scc4 script]$
sub ex7_reflection_out {  
    my ($in_dir, $in_file, $out_dir, $out_file) = @_;  

    # print out head info to mark the example 7:  
    print "IN Example 7: ";  
    print "\n";  

    open IN, "<$in_dir$in_file" or die "Sorry, can't open file ". $in_file . ".\n";  
    open input file handler to read the input  
    my $ov = <STDIN>;  
    chomp $ov;  
    if($ov eq "N") {  
        print "Please rerun the command with different output file name. ";  
        print "\n";  
        exit -1;  
    }  

    open OUT, ">$out_dir$out_file";  
    local $/;  
    local $_ = <$IN>;  
    close IN;  
    close OUT;  

    # now try to check if the output file is exactly same as the input: (it is supposed to):  
    local $y = "\n";  
    system("echo diff $out_dir$out_file $in_dir$in_file");  
    my $exit_status = system("diff $out_dir$out_file $in_dir$in_file");  
    system("echo no output is good output, exit with $exit_status");  

    print "OUT Example 7. ";  
    print "\n";  
} # end ex7_reflection_out
File IO – Example 7 output

```
[yshen16@scc4 script]$ perl fanconi_example_io.pl --example 7
IN Example 7:
WARNING - file existed, override? (Y/N)Y
diff ../data_out/fanconi_genes_dup.csv ../data_in/fanconi_genes.csv
no output is good output, exit with 0
OUT Example 7.

[yshen16@scc4 script]$  
```
Learn Through Example - RegEx
fanconi_example_str_process.pl
RegEx – String Process Example 1 code

```perl
# Example 1: convert the input comma separated data # file to be into 'tab' separated (delimited) data #

sub exl_cvs2tab {
    my($in_dir, $in_file, $out_dir, $out_file) = @_;

    # print out head info to mark the example 1:
    print "IN Example 1:");
    print "\n";

    # open input file handler
    open IN, "<$in_dir$in_file"
        or die "Can't open file $in_file to read !";
    open OUT, ">$out_dir$out_file"
        while(<IN>) {
        s/,/\t/g;
        print OUT; # Note, here the default variable '_$_' is omitted
    }
    close IN;
    close OUT;

    print "file converted, please check $out_dir$out_file";
    print "\n";
    print "OUT Example 1.";
    print "\n";
}
```

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
RegEx – String Process Example 1 output

```bash
[yshen16@sc44 script]$ perl fanconi_example_str_process.pl --example 1
IN Example 1:
file converted, please check ../data_out/fanconi_genes.txt
OUT Example 1.
[yshen16@sc44 script]$ cat ../data_out/fanconi_genes.txt
entrez_gene_symbol    entrez_gene_id
FANCA  2175
FANCB  2187
FANCC  2176
FANCD1  675
FANCD2  2177
FANCE  2178
FANCF  2188
FANCG  2189
FANCI  55215
FANCJ  83990
FANCL  55120
FANCM  57697
FANCN  79728
FANCO  5889
FANCP  84464
[yshen16@sc44 script]$  
```
sub ex2_pick_gene {
    my($in_dir, $in_file, $out_dir, $out_file, $pattern) = @_;

    # print out head info to mark the example 2:
    print "IN Example 2: ";
    print "\n";

    # open input file handler
    open IN, "<$in_dir$in_file"
      or die "Can't open file $in_file to read!";
    open OUT, ">$out_dir$out_file"
      or die "Can't open file $out_file to write!";
    <IN>;
    print OUT join ",", "entrez_gene_id, entrez_gene_name"
    print OUT "\n";
    while(my $line = <IN>) {
        chomp $line;
        my ($entrez_gene_name, $entrez_gene_id) = split ",", $line;
        if($line =~ /$pattern/) {
            print OUT join ",", $entrez_gene_id, $entrez_gene_name;
            print OUT "\n";
        }
    }
    close IN;
    close OUT;

    print "file converted, please check $out_dir$out_file";
    print "\n";
    print "OUT Example 2. ";
    print "\n";
} # end ex2_pickGenes

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
RegEx – String Process Example 2 output

[yshen16@sc4 script]$ perl fanconi_example_str_process.pl --example 2
IN Example 2:
file converted, please check ../data_out/fanconi_genes_ABCD.txt
OUT Example 2.
[yshen16@sc4 script]$ cat ../data_out/fanconi_genes_ABCD.txt
entrez_gene_id, entrez_gene_name
2175,FANCA
2187,FANCB
2176,FANCC
675,FANCD1
2177,FANCD2
[yshen16@sc4 script]$
Learn Through Example  - BioPerl

fanconi_example_gene_anno.pl
Fanconi_example_gene_anno.pl structure

1. Set up environment;
2. Open input file and get the gene list of interest
3. Initialize EntrezGene factory object
4. Call get_Stream_by_id() to fetch gene annotation info through gene id info
5. Go through the iteration of each gene
6. In each iteration, parse all the annotation attributes returned and print out result
7. Close file handle and exit.

Next, we will go through it step by step...
Load Perl on SCC

Notes, we will use Bio::DB::EntrezGene module in BioPerl suites, which is only available on SCC. So we need the two ‘module load’ commands;
Start Perl debugger by add ‘-d’ command option;

Yun Shen, Programmer Analyst
yshen16@bu.edu
IS&T Research Computing Services
Preview the Result – screen output

```
[yshen16@scc4 script]$ perl fanconi_example_gene_anno.pl
BEGIN failed--compilation aborted at fanconi_example_gene_anno.pl line 6.
[yshen16@scc4 script]$ module load perl
[yshen16@scc4 script]$ module load bioperl
[yshen16@scc4 script]$ perl fanconi_example_gene_anno.pl
IN EXAMPLE
START RUNNING EXAMPLE:
FINISHED RUNNING EXAMPLE
please check result in ./data_out/fanconi_genes_anno3.csv
OUT EXAMPLE
[yshen16@scc4 script]$ 
```
Preview the Result – how output file looks like

<table>
<thead>
<tr>
<th>entrez gene symbol: BRCA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>entrez gene id: 675</td>
</tr>
<tr>
<td>species binomial name: Homo sapiens</td>
</tr>
<tr>
<td>species common name: human</td>
</tr>
<tr>
<td>official gene symbol: BRCA2</td>
</tr>
<tr>
<td>official full name: BRCA2, DNA repair associated</td>
</tr>
<tr>
<td>cyto id: 13q13.1</td>
</tr>
<tr>
<td>Vega id: OTTHUMG00000017411</td>
</tr>
<tr>
<td>OMIM id: 600185</td>
</tr>
<tr>
<td>Ensembl id: ENSG00000139018</td>
</tr>
<tr>
<td>HGNC id: HGNC:1181</td>
</tr>
<tr>
<td>RefSeq status: REVIEWED</td>
</tr>
<tr>
<td>EntrezGene status: live</td>
</tr>
<tr>
<td>Chromosome: 13</td>
</tr>
<tr>
<td>exon count: 27</td>
</tr>
</tbody>
</table>

**Atlas symbols (total=10):**
- FAD
- FADC
- FAD1
- GLM3
- BRCC2
- FANCD
- PMCA2
- FANCD1
- XRCC1
- BROVCA2

**GO terms (total GO term = 64):**
- 278878 PMC1391BP2 (e-PCR)
- 278879 PMC1391BP3 (e-PCR)
- 278880 PMC1391BP4 (e-PCR)
- 278881 PMC1391BP5 (e-PCR)
- 278882 PMC1391BP6 (e-PCR)
- 474668 D13S1788 (e-PCR)
- 27960 STS-X05157 (e-PCR)
- 358978 REN54178 (e-PCR)
Q & A
Evaluation Please @
http://scv.bu.edu/survey/tutorial_evaluation.html

Thank You!!