

# Perl for Biologists

## Session 7

April 15, 2015

## *Regular Expressions*

Jon Zhang

# Review of Session 6

Each program has three default input/output objects associated with it

- *Input stream* – usually keyboard input: STDIN
- *Output stream* – usually to screen: STDOUT
- Error stream - usually screen: STDERR

# Review of Session 6

- Opening files for reading

```
open HANDLE, "/path/filename";
```

- Open function returns operation success

```
$res = open HANDLE, ">/path/filename";
```

# Review of Session 6

- Opening files for writing

```
open HANDLE, ">/path/filename";
```

- Opening files for appending

```
open HANDLE, ">>/path/filename";
```

# Review of Session 6

- Opened files can be read the same way as `<STDIN>`

```
$svar=<in>;
```

- Opened files should be closed when not needed

```
close(out);
```

# Review of Session 6

- Die keyword prints to STDERR

```
open HANDLE, ">filename" or die  
    "Open failed\nError is: $!";
```

- Print keyword prints to STDOUT

```
open HANDLE, ">filename" or print  
    "Open failed\nError: $!";
```

# Review of Session 6

- Buffering feature using the `$|` special variable

```
$| = 1; #don't buffer current stream
```

```
$| = 0; #do buffer current stream
```

- The concept of making a stream current

```
$prev = select(out);
```

- Using a variable for a stream handle

```
open $handle, "filename";
```


# Review of Session 6

- Reading from a binary file

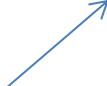
```
open HANDLE1, "/path/filename1";
```

```
binmode (HANDLE1) ;
```


```
$count = read (HANDLE1, $data , $size) ;
```



how many bytes have  
been read



binary data from file is  
stored in a variable



how many bytes to  
read



# Review of Session 6

- Reading the output stream of a program

```
open HANDLE, "program1 |";
```

- Creating a pipeline using multiple programs

```
open HANDLE, "program1 | program2 |";
```

# Review of Session 6

- Calling any program from perl

```
system("program1 arg1 arg2");
```

- Likewise, saving output

```
system("program1 arg1 arg2 1> out 2> err");
```

- Perl can run in parallel as program runs

```
system("program1 arg1 arg2 1> out 2> err &");
```

# Review of Session 6

- Opening directories

```
opendir DIRHANDLE, "/path/dirname";
```

- Reading the contents of a directory

```
@ent = readdir (DIRHANDLE);
```

- Closing directory

```
closedir DIRHANDLE;
```

# Review of Session 6

- -e “name”: file or directory exists
- -f “name”: *name* is a file
- -d “name”: *name* is a directory
- -s “name”: *name* is non-zero size
- -r “name”: *name* is readable
- -w “name”: *name* is writable
- -x “name”: *name* is executable
- -z “name”: *name* exists and has zero size

# Review of Session 6

- `mkdir("name")`: create directory *name*
- `rmdir("name")`: delete directory *name*
- `chdir("name")`: change current SCRIPT directory to *name*
- `unlink("name")`: delete file *name*
- `rename("name")`: rename file or directory *name*

# Exercise Review

- Directory `/home/jarekp/perl_06/files` contains a set of fastq files with short reads. Write a script that lists all the files in this directory.

# Exercise Review

```
opendir DIR, "/home/jarekp/perl_06/files";

foreach $entry (readdir DIR)
{
    $fullentry = "/home/jarekp/perl_06/files/$entry";
    if($entry ne "." && $entry ne "..")
    {
        print "$fullentry\n";
    }
}
```

# Exercise Review

- Modify the script from exercise 1 to open each file, read it, and produce a hash containing the distribution of sequence lengths in ALL files. Print the distribution out in descending order to a file. Plot it in Excel (no binning).



# Exercise Review

```
if($entry ne "." && $entry ne "..")
{
    print "$fullentry\n";
    open in, $fullentry;
    while($head=<in>) #while reads the header (line 1)
    {
        $seq = <in>; #read in sequence (line 2)
        $txt2 = <in>; #read in '+' line (line 3)
        $txt3 = <in>; #read in quality score(line 4)
        $len = length($seq) - 1; #(minus one for \n)
        $seq_count{$len}++;
    }
    close(in);
}
```

# Exercise Review

```
@sorted_keys = sort {$b <=> $a} keys %seq_count;
foreach $key (@sorted_keys)
{
    print "$key $seq_count{$key}\n";
}
```

# Exercise Review

- Modify the script from exercise 2 to produce fasta file containing ALL the sequences from ALL *fastq* files.

# Exercise Review

```
opendir DIR, "/home/jarekp/perl_06/files";
open out, ">sequences.fasta";
foreach $entry (readdir DIR)
{
    $fullentry = "/home/jarekp/perl_06/files/$entry";
    if($entry ne "." && $entry ne "..")
    {
        .....
    }
}
close(out);
```

# Exercise Review

```
if($entry ne "." && $entry ne "..")
{
    print "$fullentry\n";
    open in, $fullentry;
    while($head=<in>) #while reads the header (line 1)
    {
        $seq = <in>; #read in sequence (line 2)
        $txt2 = <in>; #read in '+' line (line 3)
        $txt3 = <in>; #read in quality score line (line 4)
        print out ">$head"; #no need for \n
        print out $seq; #no need for \n
    }
    close(in);
}
```

# What is a Regular Expression?

- Regex
- A specific pattern that is used to match strings of text
- Not unique to Perl
- Provides flexibility and precision in matches
- VERY applicable to bioinformatics

# What is a Regular Expression?

- We have looked at a pattern before:

```
$string = "Hello World!";  
@string_array = split / /, $string;
```

- Using simple patterns: /pattern/ and \$\_

```
$_ = "Hello World!";  
if (/Hello/)  
{  
    print "$_ contains the word Hello!\n";  
}
```

# What is a Regular Expression?

- Variables can also be used between the //

```
$match = "Hello";  
$_ = "Hello World!";  
if (/ $match /)  
{  
    print "$_ contains the word Hello!\n";  
}
```



# Binding Operators

- The binding operator: =~

```
$string = "Hello World!";  
if ($string =~ /Hello/)
```

- The other binding operator: !~

```
if ($string !~ /Bye/)
```

# Metacharacters

- Any character that does not represent itself
- `/./`: matches all but newline

```
if ($string =~ /Hel.o/)
```

- `/a|b/`: matches a OR b

```
if ($string =~ /Heli|lo/)
```

# Quantifiers

- Represents repeated instances of the preceding character

```
$string = "Hellooooo Woوووorrld!";
```

- `/a*/`: zero or more

```
if ($string =~ /Hel*i*o*/)
```

# Quantifiers

- `/a+/:` one or more

```
if ($string =~ /Hel+o+o/)
```

- `/a?/:` zero or one (i.e. optional)

```
if ($string =~ /He?a?l?lo/)
```

# General Quantifiers

- `/a{m}/`: exactly  $m$  repetitions

```
if ($string =~ /Hel{2}o{5}/)
```

- `/a{m,}/`: at least  $m$  repetitions

```
if ($string =~ /Hel{1,}o{3,}o/)
```

- `/a{m,n}/`: at least  $m$ , at most  $n$  repetitions

```
if ($string =~ /Hel{1,5}o{1,10}/)
```

# Character Classes

- Using [ ] to represent a set of characters

```
$string = "Hello World!";
```

- /[aeiouy]/: lowercase vowels

```
if ($string =~ /H[aeiouy]ll[aeiouy]/)
```

- /[012345]/: first five numbers, same as /[0-5]/

```
$string = "Hell0 World!";
```

```
if ($string =~ /Hell[0-5]/)
```

# Character Classes

- Negated Character Class using the caret ^

```
$string = "Hello World!";
```

- `/[^aeiouy]/`: anything except lowercase vowel

```
if ($string =~ /He[^aeiouy]+o/)
```

- `/[^0-5]/`: anything except first five numbers

```
if ($string =~ /Hell[^0-5]/)
```

# Character Class Shortcuts

- `/\d/`: Digit, `/[0-9]/`
- `/\D/`: Nondigit, `/[^0-9]/`
- `/\s/`: Whitespace, `/[ \t\n\r\f]/`
- `/\S/`: Nonwhitespace, `/[^ \t\n\r\f]/`
- `/\w/`: Word character, `/[a-zA-Z0-9_]/`
- `/\W/`: Nonword character, `/[^a-zA-Z0-9_]/`



# Grouping

- Using ( ) to group many characters together as one unit

```
$string = "mississippi";
```

- /(iss){2}/: looks for two repetitions of “iss” not just “s”

```
if ($string =~ /\w([aeiouy]s+){2}/)
```

# Grouping

- Grouping saves the content in the ( ) for future use in the regex using \1, \2, \3...

```
$string = "mississippi";
```

- `/(\w)\1/`: matches two of the same word characters repeated

```
if ($string =~ /\w([aeiouy]s+)\1/)
```

- Not the same as `/(\w){2}/`!

# Grouping

- Grouping saves the content in the ( ) for future use outside of the regex using \$1, \$2, \$3...

```
$string = "Hello World!";
```

- `/(\w+)\s(\w+)/`: matches first word and second

```
if ($string =~ /(\w+)\s(\w+)/)
{
    print "The first word is $1, the second word is $2\n";
}
```

- Capture variables last until next SUCCESSFUL match

# Anchors

- Anchors force the pattern to start matching at certain point in the string

```
$string = "Hello World!";
```

- `/^A\w+/` or `/^\w+/`: force matching from start of string

```
if ($string =~ /\AWorld/)
```

- `/\w+\z/` or `/\w+\Z/` or `/\w+$/`: force matching from the end of the string

```
if ($string =~ /Hello\z/)
```

# Matching

- `/pattern/` is a shortcut for `m/pattern/`
- With the 'm' you do not have to use '/'s

```
$string = "http://cbsu.tc.cornell.edu";  
if ($string =~ /\Ahttp:\/\/\//)
```
- Pick encapsulating characters that do not appear in your pattern

```
if ($string =~ m*\Ahttp://*)
```

# Match Modifiers

- `/lO wORl/i`: case insensitive match

```
$string = "Hello World!";  
if ($string =~ /lO wORl/i)
```

- `/(\w+)/g`: global match, matches all non-overlapping instances

# Match Modifiers

- Using global match and capture groups to populate an array

```
$string = "Jon Zhang Jarek Pillardy Robert Bukowski";  
@array = ($string =~ /\w+/g);
```

- Using global match and capture groups to populate a hash

```
%hash = ($string =~ /\w+/g);
```

# Match Modifiers

- `/(\w+)\s(\w+)/x`: enable adding arbitrary whitespace, very handy for readability

```
$string = "mississippi";
```



# Match Modifiers

```
if ($string =~ /
    \w          # first letter
    (          # begin group
    [aeiouy]    # any vowel
    s+         # one or more s
    )          # end group
    {2}        # group appears twice
/x
)
```

# Substitutions

```
$string = "Hello World!";
```

- `s/\w+/replacement/`: perl's find and replace function

```
$string =~ s/\w+/substitution/;
```

- `s/\w+/replaced/g`: substitutes all matches

```
$string =~ s/\w+/substitution/g;
```

- The return value of `s///` is the number of matches

```
$matches = ($string =~ s/\w+/sub/g) ;
```

# Substitutions

```
$string = "Hello World!";
```

- `s/(\w+)/$1$1/g`: using capture groups

```
$string =~ s/(\w+)/$1$1/g;
```

- `$copy = $original =~ s/pattern/sub/r`:  
nondestructive substitutions

```
$original = "Hello World!";
```

```
$copy = ($original =~ s/world/Ithaca/ir);
```

# Regex Example

- Create a hash where the keys are unique sequences of 3 base pairs and the values are the counts of how often the key appeared in the randomly generated sequence. Print out/save to a file these keys and values
  - Capture first three base pairs, increments its count
  - Delete the first base pair and repeat the process until there are less than 3 base pairs left

# Regex Example

```
$sequence
```

```
while ($sequence =~ s/([acgt])([acgt])([acgt])/$2$3/i)  
{  
    $seq_count{$1 . $2 . $3}++;  
}
```

# Exercises

1. Using our trusty random sequence generator, create a 9000 base pair length of sequence.
2. Using regular expressions find every instance of the sequence “ATGCAT” and delete it from the sequence
3. At each deletion, save the three base pairs on each side of the “ATGCAT” creating 2 arrays, one storing preceeding and one storing the trailing 3 base pairs
4. Print out the two arrays