



Unix, Perl and BioPerl

II: Sequence Analysis with Perl

George Bell, Ph.D.
WIBR Bioinformatics and Research Computing

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

Sequence Analysis with Perl

- Introduction
- Input/output
- Variables
- Functions
- Control structures
- Comparisons
- Sample scripts

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

2

Objectives

- write, modify, and run simple Perl scripts
- design customized and streamlined sequence manipulation and analysis pipelines with Perl scripts

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

3

Why Perl?

- Good for text processing (sequences and data)
- Easy to learn and quick to write
- Built from good parts of lots of languages/tools
- Lots of bioinformatics tools available
- Open source: free for Unix, PC, and Mac
- TMTOWTDI

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

4

A first Perl program

- Create this program and call it hey.pl

```
#!/usr/local/bin/perl -w
# The Perl "Hey" program
print "What is your name? ";
chomp ($name = <STDIN>);
print "Hey, $name, welcome to the
  Bioinformatics course.\n";
```
- To run: `perl hey.pl` or
- To run: `chmod +x hey.pl`
`./hey.pl`

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

5

Perl Input/Output

- Types of input:
 - keyboard (STDIN)
 - files
- Types of output:
 - screen (STDOUT)
 - files
- Unix redirection can be very helpful
ex: `./hey.pl > hey_output.txt`

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

6

Variables

- Scalar variables start with \$

```
$numSeq = 5;          # number; no parentheses
$seqName = "GAL4";   # "string"; use parentheses
$level = -3.75;      # numbers can be decimals too
print "The level of $seqName is $level\n";
$_                  # default input variable
```

- Arrays (lists of scalar variables) start with @:

```
@genes = ("BMP2", "GATA-2", "Fez1");
@orfs = (395, 475, 431);
print "The ORF of $genes[0] is $orfs[0] nt.";
# The ORF of BMP2 is 395 nt.
```

Perl functions - a sample

```
print    tr///    closedir open    m//
chomp    mkdir    split    close    die
length   chdir    join    chmod    rename
substr   opendir  pop    uc    use
s///     readdir  push    lc    sort
```

Control Structures 1

```
if (condition)    # note that 0, "", and (undefined) are false
{
    do this; then this; ...
}
else             # optional; 'if' can be used alone; elsif also possible
{
    do this instead;
}

if ($exp >= 2)    # gene is up-regulated
{
    print "The gene $seq is up-regulated ($exp)";
}
```

Control Structures 2

```
while (condition)
{
    do this;
    then this; ...
}

while ($orfLength > 100)
{
    # Add to table
    print "$seq\t";    # "\t" = tab
    print "$orfLength\n"; # "\n" = newline
}
```

Control Structures 3

```
for (initialize; test; increment)
{
    do this; ...
}

for ($i = 0; $i <= $#seqs; $i++)
# $#seqs = index of the last element in @seqs
{ # Add elements of @seqs and @orf to table
    print "$seq[$i]\t";
    print "$orf[$i]\n";
}
```

Arithmetic & numeric comparisons

- Arithmetic operators: + - / * %
- Notation: `$i = $i + 1;` `$i += 1;` `$i++;`
- Comparisons: >, <, <=, >=, ==, !=

```
if ($num1 != $num2)
{
    print "$num1 and $num2 are different";
}
```
- Note that == is very different from =
== used as a test: `if ($num == 50)`
= used to assign a variable: `$num = 50`

String comparisons

- Choices: eq, ne

```
if ($gene1 ne $gene2)
{
    print "$gene1 and $gene2 are different";
}
else
{
    print "$gene1 and $gene2 are the same";
}
```

Multiple comparisons

- AND &&
- OR ||

```
if (($exp > 2) ||
    ($exp > 1.5 && $numExp > 10))
{
    print "Gene $gene is up-regulated";
}
```

Filehandles

To read from or write to a file in Perl, it first needs to be opened.

In general, `open(filehandle, filename);`

Filehandles can serve at least three purposes:

```
open(IN, $file);          # Open for input
open(OUT, ">$file");      # Open for output
open(OUT, ">>$file");     # Open for appending
```

Then, get data all at once `@lines = <IN>;`

or one line at a time

```
while (<IN>) {
    $line = $_; do stuff with this line;
    print OUT "This line: $line"; }
```

Embedding shell commands

- use backquotes (` `) around shell command
- example using EMBOSS to reverse-complement:
``revseq mySeq.fa mySeq_rc.fa`;`

- Capture stdout from shell command if desired
- EMBOSS qualifier “-filter” prints to stdout
`$date = `date`;`
`$rev_comp = `revseq mySeq.fa -filter`;`
`print "$date";`
`print "Reverse complement:\n$rev_comp\n";`

Programming issues

- What should it do and when is it “finished”?
- Who will be using/updating your software?
 - Reusability
 - Commenting
 - Error checking
- Development vs. execution time?
- Debugging tools: printing and commenting
- Beware of OBOB (“off-by-one bug”)

Example: patscan_batch.pl

```
#!/usr/local/bin/perl -w
# Run patscan on all seqs in a folder
$myDir = "/home/elvis/seqs";
$patFile = "/home/elvis/polyA.pat";
chdir($myDir);          # Go to $myDir
opendir(DIR, $myDir);   # Open $myDir

foreach $seqFile (sort readdir(DIR))
{
    if ($seqFile =~ /\.fa$/)      # if file ends in .fa
    {
        print "Processing $seqFile\n";
        $outFile = $seqFile;      # Create $outFile name
        $outFile =~ s/\.fa\.out/;  # s/old/new/;
        ##### Run PATSCAN #####
        `scan_for_matches $patFile < $seqFile > patscan/$outFile`;
    }
}
```

Example: oligo analysis



sample fasta sequence:

```
>gi|16493450|gb|BB659629.1|BB659629
GCCTGCTTGAGTTTTGAAGTCTTGGAGCCACAGAA
AGCACTGGCCAGAGGAGAGGTAATCACTTCTAATG
CCAGGCCCTGCTGTGCAGTGGCATGTGTGATCTCA
GTCTGCTTCTGCCCTAGCTAATGAAGGCATGGACA
ATGGAATAGCCACATGGCAGCACCGGAAAACAAGC
TTACTTCTGCAGTACACAGCCTGCTTTGCTGATT
TCTGTCCACTGG
```

19

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

Basic steps for oligos.pl

Open fasta sequence

Get raw sequence

Extract oligos

Analyze oligos

Print out results

(Modify script to analyze multiple seqs)

20

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

oligos.pl: part 1

```
#!/usr/local/bin/perl -w
# Extract oligos from a sequence and analyze %GC
$seqFile = "mySeq.fa"; # input sequence
$mer = 35; # length of oligo to make
$start = 5; # nt to start oligos
$end = 11; # nt to stop oligos

# Get continuous sequence from sequence file
open (SEQ, $seqFile) || die "cannot open $seqFile: $!";
@seq = <SEQ>; # make array of lines

$define = $seq[0]; # get define
$seq[0] = ""; # delete define
$seq = join ("", @seq); # join($glue, @list)
$seq =~ s/\s*/g; # delete whitespace
```

21

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

oligos.pl: part 2

```
$seqLength = length ($seq);
print "Oligos ($mer mers) for $define
($seqLength nt) and % GC content\n";

# Beware of OBOB
for ($i = $start - 1; $i < $end - 1; $i++)
{
    # oligo = substr(seq, start, length);
    $_ = substr($seq, $i, $mer);
    $nt = $i + 1;
    $numGC = tr/GC//; # count GCs
    $pcGC = 100 * $numGC / $mer; # find %GC
    print "$nt\t$_\t$pcGC\n";
}
```

22

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

Summary

- Input/output
- Variables
- Functions (scalars and arrays)
- Control structures
- Comparisons
- Sample scripts:
 - patscan_batch.pl
 - oligos.pl

23

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

For more information, books:

- Learning Perl (Schwartz et al.) - O'Reilly
- The Perl CD Bookshelf - O'Reilly
- Beginning Perl for Bioinformatics – Tisdall
- Using Perl to Facilitate Biological Analysis (Stein) in *Bioinformatics* (Baxevanis & Ouellette)

AND several good web sites (see course page)

24

Unix, Perl, and BioPerl © Whitehead Institute, February 2004

Demo scripts on the web site

- hey.pl
- input and output options
- patscan_batch.pl
- rev_comp.pl
- oligos.pl
- parse_genbank.pl

Exercises

- Retrieving and aligning a list of human-mouse orthologs
- Retrieving a set of genes encoding growth factors, extracting their proximal promoters, and analyzing them.