Around the perl in 11 days

Where do we go from here?
Things to look into on your own.
Fancy References

• You can make complicated data structures by making lists or hashes of references to other things

• `[ .... ]` returns an reference to an unnamed list, `{ ... }` an unnamed hash
  
  – A list of hashes:

  ```perl
  @employees = ( { 'name' => 'bob', 'phone' => '555-5555' },
                   { 'name' => 'joe', 'phone' => '55555555' } );
  
  print "$employees[0]{'name'}\n";  # bob
  ```
Powerful Modules

- **DBI**
  - A simple way to access relational databases

- **CGI**
  - Create 'Common Gateway Interface' programs
  - These are programs behind the web forms like you fill out on Google, Amazon, etc.

- **GD**
  - Draw graphics, charts, plots, etc.

- There are hundreds more ...
Lazy Parsing with XML

- XML, the “eXtensible Markup Language” is a way to specify data files that are easily read by people and computers alike.
  - Many bioinformatics websites and programs provide their data in XML
- There are many Perl modules that can take care of parsing XML for you.
  - XML::Simple and XML::LibXML are good.
  - The data is transferred into a fancy data structure.
  - You need only ask for the parts you want.
    ```perl
    print "\n\n    \n\n    print "$data{ 'PubMedArticle'}{ 'Author'}\n    ```;
The Graphical Debugger

- CPAN provides a very nice graphical debugger
  - if you have the Tk and Devel::ptkdb modules you can use it simply by modifying the first line of your perl script
    
    ```perl
    #!/usr/bin/perl -d:ptkdb
    ```

- The debugger lets you walk through your script line by line and look at the contents of all of your variables

- The debugger is great for making sure your fancy data structures are correct
Adding Command Line Options

• UNIX commands often have options
  `ls -l`, etc.

• You can add options to your program using one of the getopt modules

```perl
#!/usr/bin/perl
use Getopt::Long;
GetOptions("v" => \$verbose, # Verbose mode
  "file=s" => \$file, # provide a filename
  "pvalue=f" => \$pvalue, # specify a pvalue
);```

• Run your program:
  ```bash
  foo.pl -v -file bob.txt -pvalue 0.05
  ```
Command Line perl

- You can do fancy perl commands without ever writing a script
- perl takes commands straight from the UNIX command line using a few options:
  - perl -e 'print "bob\n";'  # -e for 'execute'
  - perl -ne 'print $_;' data.txt   # -n for 'on every line of data.txt'
  - perl -pe 'tr/ACGT/acgt/;' data.txt  # -p for 'print result of -e for every line'
  - perl -nae 'print "$F[0]\n";'  # -a for 'auto split into @F on white space'
  - perl -na -F"\t" -e 'print "$F[0]\n";'  # -F for 'auto split into @F using \t character as split field
Perldoc: Document Your Scripts

• We have used perldoc to read the documentation supplied for modules

• You can add your own perldoc to your scripts by adding a few lines to it

```text
=head1 NAME
=head1 SYNOPSIS
=head1 DESCRIPTION
=head1 OPTIONS
=head1 EXAMPLES
=head1 COPYRIGHT
=head1 AUTHOR
=head1 SEE ALSO
=cut
```

• Then just fill in the blanks

• Alt-x ps in emacs will print this using the .emacs file I gave you