Introduction to Perl

March 8, 2011
by
Benjamin J. Lynch

http://msi.umn.edu/~blynch/tutorial/perl.pdf

Outline

- What Perl Is
- When Perl Should Be used
- Basic Syntax
- Examples and Hands-on Practice
- More built-in functions
- Useful Tools to Manage Your Code

Perl Is

- a very useful scripting language
- a language with many modules available to execute specific tasks in various scientific fields
- a language used to generate web content and interact with databases
- a language used to parse output text and organize results
- a language used to “glue” programs together

Perl Is

- an interpreted language
  - Perl scripts/programs do not need to be compiled as a separate step.
  - After you write a Perl script, you can immediately have a Perl interpreter execute the script.

Perl Is

- A language with minimal syntactic limitations
  - A long Perl script can be written on a single line
  - There is no required indentation for control structures
  - Perl has dynamically-typed variables, so a **string** can turn into an **integer** on the fly.

Running a Basic Perl Script

- Log in
- Open a text editor like vi, pico, or gedit
- Enter the 3 lines below and save the file as hello.pl

```
#!/usr/local/bin/perl
use strict;
print “Hello \n”;
```

Running a Basic Perl Script

- chmod +x hello.pl
- ./hello.pl

```
Hello
```

Where Should Perl Be Used?

- For problems you might solve with a shell script
- Web applications
- For problems where a Perl module offers a simple solution
  - BioPerl is a Perl library with pre-made tools for bioinformatics
  - Math::Matrix is a Perl module for matrix operations
  - http://cpan.org has large collection of these Perl modules

Problems Ill-suited for Perl

1. Executing new high-performance algorithms
   - Perl, like most interpreted languages, is slower and uses more memory
   - Fortran and C have better performance

Our Basic Perl Script

location of Perl interpreter

```
#!/usr/local/bin/perl
use strict;
print “Hello \n”;
```

tells Perl that we want to enforce a stricter syntax than the default

use the print function in Perl to print a single string

Perl Variables

- Perl has 3 categories or “contexts” of variables
  - scalar
  - list
  - hash

Scalar Context

```
my $name = 'Matt';
my $floating_point = 3.50;
my $integer_variable = 303;
```

A scalar may be a string, a 64-bit floating-point number, or an integer. The **type** will depend on value it is assigned in a particular part of your program.

my Code

```
my $name = 'Matt';
my $floating_point = 3.50;
my $integer_variable = 303;
```

For the purpose of this tutorial, place **my** before a variable name the **first time** you use it. **my** declares the **scope** for the variable that follows.

List Context

```
my @names = ( 'Larry', 'Moe', 'Curly' );

my @numbers = (13, 21, 34, 55);

my @more_numbers = (1, 1, 2, 3, 5, 8, 13,
                    21, 34, 55);
```

A single statement can span multiple lines

Hash Context

```
%lunch_menu = ( 'monday' => 'pizza',
                'tuesday' => 'burritos' );

%lunch_menu = ( 'monday' , 'pizza',
                'tuesday' , 'burritos' );
```

These are two allowed ways to assign values to a hash. A hash is a list of **key-value pairs**. A hash is also called an **associative array**.

Dynamic Types

```
my $cash = 350;

$cash = 33.58;

$cash = 'kopecks'
```

Our variable will be stored as an integer

Now it's a floating-point

Now it's a character string

Note: The first time $cash appears, we define the scope.

print

- The print function will print the contents of a scalar or a list.
- by default, it will print to STDOUT (the screen) unless you also give it a FILEHANDLE.

```
print @names;
```

If you're interested in formatted printing,
check out the **write**, **printf**, and **sprintf** functions

Perl Math

- Perl also has standard math functions

```
$sum = 2 + 3;
print cos(0.0);
$fraction = 17/42;
$product = 19*23;
$four = sqrt(16);
$nine = 3**2;
$difference = 9-2;
print 'in 3rd grade, 14/3 was 4 remainder ', 14%3;
```

If an integer gets too large, it will dynamically be changed into a floating-point number

Exercise 1

http://www.msi.umn.edu/tutorial/scientificcomp/perl2/

- Assignment - What to do
- Details - further explanation
- Tips - tips to speed up your progress
- Solution - one of many solutions

More Operators

- Perl also has functions and operators for string concatenation, manipulating lists, incrementing integers, etc.

```
$a++;   (the $a variable will increase by 1)
$a+=3;   (the $a variable will increase by 3)
$a*=2;   (the $a variable will be multiplied by 2)
$a/=7;   (the $a variable will be divided by 7)
$a--;   (the $a variable will decrease by 1)

$string = 'words';
$b = 'Words, '.$string;
$b.=', words';
```

Words, words, words

Control Structure

- Perl has control structure elements that are very similar to other languages. Some examples include:

```
if ( $my_variable > 42 ) {
    do_this();
}

while ( $num < 42 ) {
    $num++;
}
```

Control Structure

- 2 types of for loops

```
for (1 .. 10) {
    print $_, "\n";
}
```

$_ is a special variable in Perl. Here, it holds the next value in our list.

```
foreach my $item (@some_list) {
    print $item, "\n";
}
```

Subroutines

- Perl Subroutines are chunks of code that have a list passed to them, and they return a result.

@_ is a special variable in Perl that holds the list of items passed into a subroutine

```
print product (8,4);

sub product {
   my $product=1;
   foreach my $value (@_){
      $product *= $value
   }
   return $product
}
```

Exercise 2

http://www.msi.umn.edu/tutorial/scientificcomp/perl2/

- Assignment - What to do
- Details - further explanation
- Tips - tips to speed up your progress
- Solution - one of many solutions

Regular Expressions

- Regular expressions can be used in Perl to test conditions, search output files, or perform a search/replace.

```
if ( $string =~ /(some pattern)/ ){
   do_something();
}
```

```
$string =~ s/search/replace/ ;
```

Special Variables

- Perl has dozens of special variables. Many of them should be avoided. Here's a few useful ones.

@ARGV	The list of arguments passed when you executed the program.
@_	The list of items passed to a subroutine.
$_	The current topic.
$$	The process ID of your perl script

Context

- If you want a single value from a list, you need to remember to use the scalar context

```
@names=('chuck','larry');
print $names[0];
```

```
chuck
```

Context

- The context used can greatly change the behavior of an operator

```
@names=('chuck','larry');
$num = @names;
print $num;
```

2

```
@matches = ($string =~ /pattern/g);
```

Because we used the "g" global option with this expression, we expect to get multiple matches (and so does the assignment operator).

Files

- When you open a file in Perl, you specify a FILEHANDLE. This is a name to identify the access to the file in your program.

The name “INPUT” is the FILEHANDLE.

“>” == write access
“<” == read access
“>>” == append access
“+>” == read/write

```
open INPUT , “<” , “file.txt”;

open INPUT , “>” , “file.txt”;

open INPUT , “>>” , “file.txt”;

open INPUT , “+>” , “file.txt”;

close INPUT;
```

Reading and Writing

- The File Handle is used in a print statement when you want the result to be sent to a file instead of the screen.

```
open INPUT ,  “<” , “file.txt”;

while (<INPUT>){
  print $_;
}
```

```
open OUTPUT ,  “>” , “out.txt”;
print OUTPUT “Hello World”;
```

Exercise 3

http://www.msi.umn.edu/tutorial/scientificcomp/perl2/

- Assignment - What to do
- Details - further explanation
- Tips - tips to speed up your progress
- Solution - one of many solutions

Modules

- Many modules exist to solve common problems. These include general math functions, file parsers for common formats, as well as tools for very specific applications.

Each module will have its own functions and special variables. Read about any module you're interested in to see how it works.

```
use Math::Matrix;

$a = new Math::Matrix (
                [1, 3, 4],
                [2, 6, 4],
                [2, 3, 7]);
$z = $a*$a;
```

Tools to Keep Your Code Clean

- The flexibility of Perl can make the code difficult to read
  - built-in functions like “open” can be executed in multiple ways
  - indentation is not enforced in control structure
- The flexibility of Perl can make your code difficult or even dangerous to re-use
- Tools like perltidy and Perl::Critic can help!

perltidy

- available on SDVL machines
- or download it at: http://perltidy.sourceforge.net
- It reformats your Perl code to indent control structures, and to use a consistent scheme for spacing.
- It can be run like this:
  - perltidy < mycode.pl > mycleancode.pl

What does perltidy do?

```
use strict;
for my $i (0 .. 3){for my $j (0 .. 3){for my $k (0 .. 3){
my $sum=$i+$j+$k; print $sum. "\n" }}}
```

```
use strict;
for my $i ( 0 .. 3 ) {
    for my $j ( 0 .. 3 ) {
        for my $k ( 0 .. 3 ) {
            my $sum = $i + $j + $k;
            print $sum. "\n";
        }
    }
}
```

Perl::Critic

```
/usr/local/bin/perl -MPerl::Critic=critique -e 'print critique(shift)' code.pl
```

- Perl::Critic will warn you about unwise programming style
- It has 5 levels of warnings to warn you about problems ranging from dangerous syntax to unreadable expressions.

Thanks for Coming!

Send Questions To:

- help@msi.umn.edu
- blynch@umn.edu