Perl

- If
  - just as you may have guessed :)
  - just like in C we could used !=, <, or >

```
$i = 0;
if ( $i == 0 ) {
   print "it's true\n";
} else {
   print "it's false\n";
}
```

Perl

- While Loops
  - execute statements if condition is true

```
while ( $i == 0 ) {
    print "it's true\n";
    ...
    <do some things that may modify the value of $i>
    ...
}
```

Perl

- For Loops
  - just like in C
  - (*from, to, increment*)

```
for ($i = 0 ; $i < 10 ; $i++ ) {
   print $i, " ";
}
print "\n";
```

counts from 0 to 9 and prints the value (without a newline until the end) and generates:

```
0 1 2 3 4 5 6 7 8 9
```

Perl

- Foreach Loops
  - just like in C shell

A foreach loop looks like this:

```
foreach $n (1..15) {
   print $n, " ";
}
print "\n";
```

and generates about what you would expect:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Perl

- File I/O
  - unlike shell scripts no we can input and output from/to files, rather than only stdin stdout.
  - You still can access standard input and output:

```
while (@line=<stdin>) {
  foreach $i (@line) {
     print "->", $i;        # also reads in EOL
    }
}
```

This script will read each line from the standard input and print it.

Perl

- File I/O
  - Assume you have a specific data file you wish to read from:

```
$FILE="info.dat";
open (FILE);              # name of var, not eval
@array = <FILE>;
close (FILE);
foreach $line (@array) {
   print "$line";
}
```

This Perl script opens "info.dat" and reads all its lines into the array called "array". It then prints out each line.

Perl

- Functions
  - Perl functions are simple to use, although the syntax can get complicated.

```
sub pounds2dollars {
    $EXCHANGE_RATE = 1.54;  # modify when necessary
    $pounds = $_[0];
    return ($EXCHANGE_RATE * $pounds);
}
```

  - This function changes a value specified in pounds sterling into US dollars (exchange rate of $1.54 to the pound, which can be modified as necessary). The special variable $_[0] references the first argument to the function.

Perl

- Functions

```
sub pounds2dollars {
   $EXCHANGE_RATE = 1.54;  # modify when necessary
   $pounds = $_[0];
   return ($EXCHANGE_RATE * $pounds);
}
```

  - To call the function, our Perl script would look like this:

```
$book = 3.0;              # price in British pounds
$value = pounds2dollars($book);
print "Value in dollars = $value\n";
```

When we run this script (which includes the Perl function at the end), we get:

```
Value in dollars = 4.62
```

Perl

- Library Functions
  - Perl has ability of Linux system calls via Perl library functions
  - we have already used *open() cose(),* and *print()*

```
exit(1);    # exit a Perl program and pass the
            # specified return code to the shell.
```

  - Perl also provides a special exit function to print a message to stdout and exit with the current error code:

```
open(FILE) or die("Cannot open file.");
```

if call to *open*() fails the *die*() function is executed

Perl

- Library Functions
  - Some string functions to assist in manipulating string values are length(), index(), and split():

```
$len = length($fullname);
```

sets the $len variable to the length of the text stored in the string variable $fullname.

Perl

- Library Functions
  - To locate one string inside another:

```
$i = index($fullname, "Smith");
```

The value of $i will be zero if the string begins with the text you specify as the search string (the second argument).

Perl

- Library Functions
  - To divide up a line of text based on a delimiting character (for example, if you want to separate the tokens from the Linux password file into its various parts):

```
($username, $password, $uid, $gid, $name, $home, $shell)
                                = split(/:/, $line)
```

In this case, the split() function returns an array of values found in the string specified by $line and separated by a colon.

Perl

- Library Functions
  - Another common function provides your Perl program with the time and date:

```
($s, $m, $h, $dy, $mo, $yr, $wd, $yd, $dst) = gmtime();
$mo++;                          # month begins counting at zero
$yr+=1900;                      # Perl returns years since 1900
print "The date is $mo/$dy/$yr.\n";
print "The time is $h:$m:$s.\n";
```

The code above produces the following result:

```
The date is 10/07/2010.
The time is 18:40:27.
```

Note that gmtime() returns 9 values. The Perl syntax is to specify these values in parentheses (as you would if you were assigning multiple values to an array).

Perl

- Command-Line Arguments
  - we can pass command-line arguments to a Perl script

```
$n = $#ARGV+1;  # number of arguments (beginning at zero)
print $n, " args: \n";
for ( $i = 0 ; $i < $n ; $i++ ) {
   print "   @ARGV[$i]\n";
}
```

This Perl script prints the number of arguments that were supplied on the **perl** command (after the name of the Perl script itself) and then prints out each argument on a separate line.

Perl

- Command-Line Arguments
  - modified British pound conversion

```
if ( $#ARGV < 0 ) {  # if no argument given
   print "Specify value in to convert to dollars\n";
   exit
}
$poundvalue = @ARGV[0];     # get value from command line

$dollarvalue = pounds2dollars($poundvalue);

print "Value in dollars = $dollarvalue\n";

sub pounds2dollars {
   $EXCHANGE_RATE = 1.54;   # modify when necessary

   $pounds = $_[0];
   return ($EXCHANGE_RATE * $pounds);
}
```