Perl

- Practical Extraction and Reporting Language
  - general-purpose, high level, general-purpose, interpreted, dynamic programming language
  - invented by Larry Wall 1987 (a linguist working at NASA)
  - different versions of Perl, upcoming: Perl 6
  - links:
    - www.perl.org/ general site
    - http://perldoc.perl.org/index-tutorials.html tutorials

Running Perl

- *perl [-c] fileName*
  - -c argument only checks for syntax but does not execute the script
- *perl -v*

-bash-3.2$ perl -v
This is perl, v5.8.8 built for x86_64-linux-thread-multi
Copyright 1987-2006, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on this system using "man perl" or "perldoc perl". If you have access to the Internet, point your browser at http://www.perl.org/, the Perl Home Page.

Perl

- Running a Perl script
  - *-bash-3.2$ perl file.pl*
  - or use #! in first line of script
    - #!/usr/bin/perl
- perl version of “hello world”
  - *print “hello world. \n”*; each line must end with “;”

Perl

- Simple Variables
  - variables always use $ sign, e.g.,
  - $i =3;

Perl

- Variables, Strings and Integers
  - strings specified by text in quotation marks
  - strings can be concatenated by operator “.”
  - integers support a range operator, e.g., 3..15

```
print 1, 2, 3..15, "\n";     # range operator
print "A", "B", "C", "\n";   # strings
$i = "A" . "B" ;             # concatenation operator
print "$i", "\n" ;
```

  - output

```
123456789101112131415
ABC
AB
```

Perl

- Arrays
  - dynamic allocation (don't need to worry about allocation, it is done for you)
  - arrays use @ symbol, e.g., @arr

```
@arr = (1,2,3,4,5);
```

This line defines the array "arr" and puts 5 values in it. Same as

```
@arr = (1..5);
```

```
print @arr[0],"\n";
```
prints out first element of arr

Perl

- Arrays
  - array elements start with 0
  - use array index to access specific element
  - if only array name is printed, the entire array is printed
  - using an array in a scalar operation will be interpreted as the number of elements in the array

Perl

- Arrays

```
@a1 = (1);              # array of 1 element
@a2 = (1,2,3,4,5);      # array of 5 elements
@a3 = (1..10);          # array of 10 elements

print @a1, " ", @a2, " ", @a3, "\n";

print @a1[0], " ", @a2[1], " ", @a3[2], "\n";

# using as scalar will yield number of items
print @a2 + @a3, "\n";
```

will result in the following output:

```
1 12345 12345678910
1 2 3
15
```

Perl

- Associated arrays
  - rather than using index with value 0 to maximum size of array the array value can be used to access elements

```
@month{'January'} = 1;
@month{'February'} = 2; ...
```

and so on. Then you can read in the month name and access its numeric value this way:

```
$monthnum = $month{$monthname};
```

Perl

- alternative way to set up array

```
%month = ("January", 1, "February", 2, "March", 3,
          "April", 4, "May", 5, "June", 6,
          "July", 7, "August", 8, "September", 9,
          "October", 10, "November", 11, "December", 12);
```

The set of values that can be used in an associative array, or the keys to the array, are returned as a regular array by a call to the Perl function **keys()**:

```
@monthnames = keys(%month);
```

Perl

- Mathematical and Logical Operators
  - similar to other languages
  - +, -, *, /
  - integer increments before or after value is used

Perl

- examples

```
$n = 2;
print ("\$n=", $n, "\n");

$n = 2 ; print ("increment after \$n=", $n++, "\n");
$n = 2 ; print ("increment before \$n=", ++$n, "\n");
$n = 2 ; print ("decrement after \$n=", $n--, "\n");
$n = 2 ; print ("decrement before \$n=", --$n, "\n");
```

This script generates the following output:

```
$n=2
increment after $n=2
increment before $n=3
decrement after $n=2
decrement before $n=1
```

Perl

- examples

```
$n = 2;
print ("\$n+2=", $n + 2, "\n");
print ("\$n-2=", $n - 2, "\n");
print ("\$n*2=", $n * 2, "\n");
print ("\$n/2=", $n / 2, "\n");
```

This script generates the following output:

```
$n+2=4
$n-2=0
$n*2=4
$n/2=1
```

Perl

- examples

```
$r = 3.14;                          # real number
print ("\$r=", $r, "\n");

print ("\$r*2=", $r * 2, "\n"); # double
print ("\$r/2=", $r / 2, "\n"); # cut in half

print ("1 && 1 -> ", 1 && 1, "\n");
print ("1 && 0 -> ", 1 && 0, "\n");
print ("1 || 1 -> ", 1 || 1, "\n");
print ("1 || 0 -> ", 1 || 0, "\n");
```

This script generates the following output:

```
$r=3.14
$r*2=6.28
$r/2=1.57
1 && 1 -> 1
1 && 0 -> 0
1 || 1 -> 1
1 || 0 -> 1
```

Perl

- String Operators
  - only simple operation is concatination

```
$firstname = "Bob";
$lastname = "Smith";
$fullname = $firstname . " " . $lastname;
print "$fullname\n";
```

results in the output:

```
Bob Smith
```

Perl

- String Operators
  - several simple matching operations are available

```
if ($value =~ /abc/) { print "contains 'abc'\n"};
$value =~ s/abc/def/;    # change 'abc' to 'def'
$value =~ tr/a-z/A-Z/;   # translate to upper case
```

Perl

- Comparison Operators

Figure 4-34. Perl comparison operators.

Operation	Numeric values	String values
Equal to	==	eq
Not equal to	!=	ne
Greater than	>	gt
Greater than or equal to	>=	ge
Less than	<	lt
Less than or equal to	<=	le