

Perl

- **Allegedly an acronym for *Practical Extraction and Reporting Language* or *Pathologically Eclectic Rubbish Lister*, actually a variant of “pearl.”**
- **Invented by Larry Wall.**
- **Improvement upon *awk* and *sed*.**
- **Ported to most desktops platforms, widely used for Internet applications, but not limited to it.**
- **Easy to extend with modules -- a variety may be found at <http://www.cpan.org/> (Comprehensive Perl Archive Network)**
- ***There's More Than One To Do It.***

Perl program

- Text based language
- Minimal Perl program

```
print "Hello, world\n";
```

* Would be executed using command ‘perl hello.pl’

- Unix/Linux script form

```
#!/usr/bin/perl
```

```
print "Hello, world\n";
```

* First line tells shell what interpreter to use

* Must also make script executable using *chmod*

e.g. *chmod 755 hello.pl*

Perl structure

- **#** used for comments
- Usual C-like control statements: *if*, *while*, *do*, *switch*
 - * brackets required for one line statements

```
if ($length > 20) {  
    print "A long one."  
}
```

- **Alternate form available**

```
print "value = $value" if $debug > 1;
```

- **Alternate keywords *unless* and *until* also supported in place of if and while.**
- **Alternate loop construct *foreach* retrieves each element of a collection**

```
foreach $word (@somelist) {
```

- ***next* and *last* alternate loop flow**
 - * **next skips rest of loop and does next iteration (analogous to C continue)**
 - * **last exits loop (analogous to C break)**
 - * **can use with labels to leave nested loops**

```
OUTER: for ($x=0;$x<SIZE;$x++) {  
  for ($y=0;$y<SIZE;$y++) {  
    last OUTER if $matrix[$x][$y] == 4;  
  }  
}
```

Perl operators

- Most of the usual C-like operators
- `==` , `!=`, `<`, `>`, `<=`, `>=` are numeric
 - * Use *eq*, *ne*, *lt*, *gt*, *le*, *ge* for strings.
- `&&` and `||` are high precedence and short circuit
 - * *or* and *and* are lower precedence alternatives

```
my $file="myfile.txt";
```

```
open $file or croak "Can't open $file $!\n";
```

- `!` is high precedence logical negation
- *not* is low precedence logical negation

Perl variables

- Perl is loosely typed -- same variable can hold a integer, real or string.
- Three primary data types: scalars, arrays, and hashes (AKA associative arrays)
- Perl handles allocation automatically
 - * Unassigned variables have special *undefined* value
 - * *undef* command removes assignment from variable
- *scalar* is any single thing -- may be integer, floating point, string, et. al.
 - * Indicated by a leading \$ e.g. \$total

- **Strings are a series of characters**
 - * **Can use either single or double quotes to declare**
 - * **Single quotes put exactly what is inside into the string**
 - * **Double quotes substitute variables and special characters**
 - \n is newline**
 - \t is tab**
 - * **Use . to concatenate strings**
 - * **Many built in functions available to manipulate strings**

- **an *array* is a linear, indexed collection of elements.**
 - * **first index is zero**
 - * **Indicated by a leading @ e.g. @names**
 - * **Element accessed by square brackets**

```
$names[4] = 'Gerard';
```

Note individual element is a scalar, so \$ is used.

- **a hash is a collection indexed by a key**
 - * **Indicated by a % - e.g. %phoneNumbers**
 - * **Element accessed/inserted by a key value**

```
print $phoneNumbers{'Gerard'};
```

- * ***exists* tests for existence of key in hash**
- * ***delete* removes key-value pair from hash**
- * ***keys* returns list of keys in hash -- order unspecified**

- **Variables can be used without declaration -- not a good thing.**
 - * Use *my* keyword to declare and limit scope.
 - * Use *strict* directive to require declarations of variables

```
use strict;
```

```
my $name = 'Larry Wall';
```

```
print "Perl was invented by $name";
```

- * **Error with *strict* in place**

Silent runtime error without *strict*.

- **Aggregate initialization may be used for arrays and hashes**

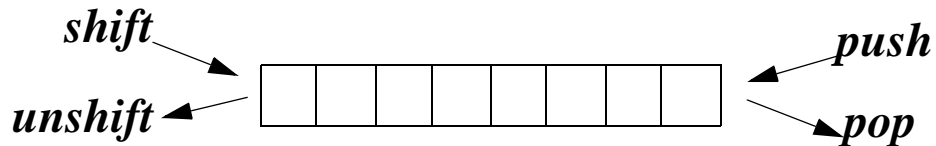
```
my @names=('Ali','Brian');
```

```
my %urls = (Hartford => 'rh', Troy => 'rpi');
```

- **If a list / hash is provided where a scalar is expected, that is, in *scalar context*, the number of elements is returned.**

Additional data structures

- **Stack, queue and deque functionality available by using arrays and perl functions *push*, *pop*, *shift* and *unshift*.**



- **References are also available and stored in scalars.**
- **Objects are also available -- handles are stored in scalars.**
 - * use *new* to create new object
 - * fairly easy to use objects
 - * syntax for creating objects differs from C++/Java

Copy an example!

Subroutines

- **Define a subroutine (function) using the *sub* keyword**

```
sub factorial {  
    my $arg = shift;  
    defined $arg or croak "arg not defined\n";  
    my $product = 1;  
    for (;$arg > 1;$arg--) {  
        $product *= $arg;  
    }  
    return $product;  
}
```

- **Arguments are passed in special `@_` array.**

- * **May be assigned to local variables using *shift*.**

```
my $arg = shift;
```

- * **Traditional code to check that arguments were passed:**

```
defined $arg or croak "arg not defined\n";
```

- **Last value processed is return value**

* *return* keyword makes this much clearer

```
$product;          #equivalent statements
```

```
return $product;  #also exits function
```

- **Can call a undeclared function with parentheses.**

```
factorial(5);
```

- **Can call a declared function without parentheses, like built-in functions**

```
sub factorial; #defined elsewhere
```

```
....
```

```
factorial 5;
```

- **Perl 5 added prototypes**

```
sub factorial($); #forward declaration
```

```
sub factorial ($){ #definition
```

* **Makes a call like *factorial*; a compile error**

* **Without it is a return error caused by *croak*.**

- **Prototypes can also use @ and % to require arrays or hashes.**
 - * **Tricky because you can pass a list of scalars to a function wanting an array**

```
sub printall {  
    my $word;  
    foreach $word (@_) {  
        print "$word\n";  
    }  
}  
my @list = qw(Rensselear at Hartford);  
printall @list;  
printall 3,4,5;
```

* *qw* quotes the individual words “Rensselear”, “at”, “Hartford”

Perl modules

- Can modularized a set of code (special type of library)
- Invoke by *use* statement

```
use strict;
```

```
use Carp;
```

- Defined in file that ends with *.pm*

- Typical module beginning

```
package Carpex;
```

```
require Exporter;
```

```
@ISA = qw(Exporter);
```

```
@EXPORT = qw(factorial demo);
```

- Client which calls

```
use Carpex;
```

will automatically have *demo* and *factorial* added to namespace

Carp module

- **Very useful standard module is *Carp***
 - * **Perl has *warn* and *die* built in**
 - warn prints error message to standard error (stderr)**
 - die prints error message to stderr and exits**
 - both tell you the line with die or warn in it**
 - * ***Carp* module adds *carp* and *croak***
 - Tell you the line that called the package routine including the carp or croak in it**
 - * ***confess* dumps out full stack trace**
- **Example**

- **Example**

```
#carpex.pl  
factorial -3; #line 5
```

```
#Carpex.pm  
sub factorial ($){  
    my $arg = shift;  
    KEYWORD "factorial argument $arg must be positive" unless $arg >0; #line 17  
    * if keyword is die, prints
```

factorial argument -3 must be positive at Carpex.pm line 17.

```
    * if keyword is croak, prints
```

factorial argument -3 must be positive at carpex.pl line 5

```
    * if keyword is confess, prints
```

factorial argument -3 must be positive at Carpex.pm line 17

```
    Carpex::factorial(-3) called at carpex.pl line 5
```

- **Hint: always use *strict* and *Carp***

File I/O

- Use *open* to open up a file

`open INFILE, "input.txt" or croak "Can't open input.txt $!";`

- * `#!` contains error message from last system command

- * by default, opened for reading

- * precede with `>` to make output file

`open OUTFILE, ">$tempName" or croak "Can't open $tempName $!";`

- * trail with `|` to execute command and read command output

- * for some reason you don't declare filehandles before using, even with *strict*

- Use angle brackets to read a line from file

`$line = <INFILE>;`

- **Use `foreach` to read each line in turn**

```
foreach $line (<INFILE>) {
```

- * **Use `chomp` to removed the newline from the end of line**

```
  chomp $line;
```

- * **Possible to read entire file in array, but that will use up memory if file is a large one.**

- **Use *print* to write to output file**

```
  print OUTFILE "$line\n";
```

- * **Note there's no comma after filehandle**

- **It's best to close file when finished with it.**

```
close INFILE;
```

Regular expressions

- Much of Perl's text processing functionality comes from *regular expressions*
 - * a way of representing patterns in words
 - * consists of things to match, cardinality, location anchors, and options
- single letters match themselves
 - * *cat* would match cat, concatenate, caterpillar, bobcat
- *.* matches a single letter
 - * *b.d* would match bad, bed, bid, bod, bud, bcd, etc

- **[] matches any of a set of things**
 - * **[aeiou] matches any vowel**
 - * **[a-z] matches any letter between a and z, inclusive**
- **| means or**
 - * ***dog/cat* matches dog, hot dog, cat, catfish, etc.**
- **Special classes**
 - * **\d matches digit, \D matches nondigit**
 - * **\w matches a word (alphanumeric), \W matches nonword**
 - * **\s matches whitespace, [\t\n\r\f], \S matches not space**

- **Cardinality**

- * * after pattern matches 0 or more

- bl*ind* matches bind, blind, bllind, etc.

- * + matches one or more

- ie+* matches brief, ieee, but not pit

- * ? matches 0 or one

- cl?ap* matches cap and clap but not catnap

- * {m,n} matches between m and n times

- * {m,} matches m or more

- * {m} matches exactly m times

- **Anchors**

- * **^ matches at beginning only**

- ^*cat* matches cat, caterpillar but not bobcat**

- * **\$ matches at end only**

- ion*\$ matches diction, ion but not ionize**

- **options**

- * **i makes case insensitive**

- * **g matches all occurrences**

- * **o only compiles pattern once**

- **To test for match in a string, use bind operator. =~**

- if (\$line =~ m/disclaimerBegin/oi) {**

- note m is assumed if no operator present**

- if (\$line =~ /disclaimerBegin/oi) { #same thing**

- **To test for a not match, use !~**
- **To substitute, use s operator**

`$ansidef =~ s/
/\n/g;`

- **Parentheses both group and *backreference***

* *c(ab)*x* matches cabx, cababx, cabababx, etc.

* values which match expressions in parantheses are stored in variables \$1, \$2, \$3, etc.

f(\w)at* matches frat, \$1 = r

f(\w)at* matches fortunate, \$1 = ortun

CGI module

- **CGI module allows use of Perl for CGI scripting**
- **Handles gets, posts, parameter passing, and error handling**
- **Methods for common HTML and form elements**
- **Object interface**
 - * **Preferred usage is to use explicit object to make clear what methods are coming from CGI module and what are builtin or from other modules.**

```
$query->start_form;
```

- **Use CGI::Carp to direct errors to browser**

```
use CGI::Carp qw/fatalsToBrowser/;
```

Taint checks

- *taint* - *to touch or affect slightly with something bad*
- **If user feeds bad data, Perl could do bad things**

open \$arg or croak "Can't open \$arg \$!"; #seems harmless

but if \$arg was user input and is *rm -fr //* command will (attempt to) delete entire file system

- **If invoked with -T option, Perl tracks data input via command line arguments, environments or data files**
- **Refuses to execute, e.g. via *exec* or *open*, until untainted**
 - * **To untaint, process through regular expression and extract values using backreferences (\$1, \$2)**
 - * **Requires developer smart enough to write good regular expression**

- **Taint invoked by -T option**
- **CGI script should normally start**

#!/usr/bin/perl -T

*** ensures taint checking turned on**

- **Perl sets taint checking automatically on Unix/Linux setuid programs**
 - * best to set explicitly if writing setuid type script**

Other Perl goodies

- Often desirable to print multiple lines of text out
 - * “Here” documents defined by `<<label`
 - * prints following lines until terminated by label

```
print<<EOM;  
<p><h1>Pattern match test form</h1></p>  
<p><a href="http://www.rh.edu/~gerardw/opensource">Open Source page</a>  
</p>  
EOM
```

- Documentation conventions
 - * Utilities to convert from perldoc to HTML, et. al.
- Integration with other languages
 - * Can link C/C++ libraries into perl
 - * Can embed Perl in C/C++ programs

- **Modules for almost anything on the net**
 - * **Reuse is better than rewrite**
- ***pl2bat* converts Perl script to windows batch file**
- **There's even a cookbook!**