

Because Laziness is a Virtue

A brief tour of the Perl language

Sean Walberg <sean@ertw.com>

January 2004

Outline

- Introduction
- The Perl Language
- CPAN
- Applications
- Further Reading

Introduction

- Practical Extraction and Reporting Language
- 1.0 Released Dec, 1987 by Larry Wall
- Powerful regular expression features
- Extensible
- Write only?

Me

- Started out with Perl 4 in 1995 to write reports and help with systems admin
- ```
#!/bin/perl -sp0777i<X+d*1MLa^*1N%0]dsXx++1M1N/dsM0<j]dsj
$/=unpack('H*',$_);$_=`echo 16dio\U$k"SK$/SM$n\EsN0p[1N*1 1K
[d2%Sa2/d0$^Ixp"|dc`is/\W//g;$_=pack('H*',/((..)*)$/)
```
- No modules or CPAN at the time
- Terse, yet expressive



# The Perl Culture/Principles

- Three great virtues of a programmer
  - Laziness
  - Impatience
  - Hubris
- There Is More Than One Way To Do It
- A language should not force a programmer to do something a certain way

# Larry Wall

- Grad degree in languages
- Created Perl to “solve a problem AWK couldn't handle”
- Also created rn and patch
- Well known for witty quotes



# The Perl Language

- Derived from C, sh, AWK, sed, and many others
- Interpreted, but compiled at runtime
- “Easy things should be easy, hard things should be possible”

Just don't compare it with a real language, or you'll be unhappy... :-)  
- Larry Wall

# Language Basics

- Lines terminated by semicolon (;)
- Comments use hash (#)
- Double quotes allow interpolation, single quotes don't
- Context is important! (we'll see what this means)
- Lots of defaults and implied variables

Break; /\* don't do magic till later \*/  
- Perl Source

# Data Types

- Scalars
- Arrays
- Hashes

# Scalars

- Integer, float, or string, prefixed with \$
- \$name = “Sean”
- \$degF = 9/5\*\$degC+32
- Concatenate with .
  - \$foo = “Bar” . “Baz”
- Interpolated: “Hello, \$name”
- Not interpolated: 'Hello, \$name'
- Force something to scalar with “scalar”

# Arrays

- Multidimensional construct, prefixed with `@`
- `@months = ("Jan", "Feb", ...)`
- `@months = qw /Jan Feb .../ # easier`
- Refer to elements in scalar context
  - `print $months[0] # NOT @months[0]!`
- In scalar context, `@months` returns the number of elements. In list context, returns all the elements!
  - `print @months vs. print scalar @months`

# Hashes (Associative Arrays)

- “Named Array”, prefixed with %
- %eyes = ( “Sean” => “Hazel”, “Jon” => “Blue” )
- Again, refer to elements in scalar context
  - print \$eyes{‘Sean’}
- IMHO, one of the more powerful features of Perl
- Iterate with: keys, values, each

Doing linear scans over an associative array is like trying to club someone to death with a loaded Uzi.

-Larry Wall

# References

- Like a pointer in C
- `$eyes = \%eyes`

Beauty? What's that?  
-Larry Wall
- `print $eyes->{'Sean'}`
- You can get some crazy data structures, such as a hash whose elements are arrays containing hashes themselves
  - `$people->{'Sean'}->[0]->{'eyes'}`

# Basic Structures

```
while ($foo) { $foo--; } # Basic, eh?
$foo-- while($foo); # Same thing!
for ($i=0; $i<100; $i++) {
 print $i."\n"; }
print "The answer!\n" if ($a == 42);
foreach $i @array { print $i."\n"; }
print map {$_."\n"} @array #TIMTOWTDI!
```

# Regular Expressions

- `=~` “Has within it”, `/ /` delimits regexps
- `print “\${a} contains perl” if (\${a} =~ /perl/);`
- `\$doc =~ s/use/utilize/ig;`
- Uses standard regular expressions, and
  - `\d` – numeric (or `\D` for non-numeric)
  - `\w` – alphanumeric (`\W` nonalpha)
  - `\s` – whitespace (`\S` nonwhitespace)

# Sorting

```
foreach $i (sort @a) { print $i } # EZ
foreach $i (reverse sort @a) ...#Reverse
foreach $i (sort { $b <=> $a } @a) #ditto
```

On the subject of sorting, when you pull data out of a hash, you can't predict the order!

```
foreach $i (sort keys(%hash)) { #key sort
 print "$i = " . $hash{$i} . "\n"; }
```

Use custom sort routine as above to sort on values such as

```
sort { $hash{$a} <=> $hash{$b} } keys(%hash)
```

# Implied Variables

```
While ($a = <FOO>) { # Read FH
 $a =~ s/blah/quux/; print $a; }
```

`$_` is implied for reads from FH's, regexp, and print

```
while (<FOO>) { # $_ instead of $a
 s/blah/quux/; print; }
```

Subs return value of last operation if no return (see sorting)

Implied variables decrease clutter, but increase complexity – use with care!

# Like You'll Ever Remember These

- `$|` - line buffering
- `$_` - general purpose
- `@_` - args passed to function
- `$>` - EUID
- A whole page in the index of the Camel book!

Even if you aren't in doubt, consider the mental welfare of the person who has to maintain the code after you, and who will probably put parens in the wrong place.

-Larry Wall

# CPAN

```
[root@poochie root]# perl -MCPAN -e shell

cpan shell -- CPAN exploration and modules installation (v1.76)
ReadLine support enabled

cpan> i /rot13/
CPAN: Storable loaded ok
Going to read /root/.cpan/Metadata
 Database was generated on Mon, 05 Jan 2004 12:52:06 GMT
Distribution A/AY/AYRNIEU/Crypt-Rot13-0.04.tar.gz
Module Crypt::Rot13 (A/AY/AYRNIEU/Crypt-Rot13-0.04.tar.gz)
Module LWP::Sink::rot13 (G/GA/GAAS/LWPng-alpha-0.24.tar.gz)
3 items found

cpan> install Crypt::Rot13
Crypt::Rot13 is up to date.
```

- Almost everything available on CPAN
- Easy to use interface, handles dependencies and versions
- Searchable, lots of mirror sites
- Often more than one module for a task, read the docs closely

# CPAN growth

- CPAN's growth shows the popularity of Perl
- Half the modules were added in 2003

| <u>Year</u> | <u>Modules Added</u> |
|-------------|----------------------|
| 1995:       | 30 ( 0.51% )         |
| 1996:       | 35 ( 0.59% )         |
| 1997:       | 68 ( 1.16% )         |
| 1998:       | 189 ( 3.21% )        |
| 1999:       | 287 ( 4.88% )        |
| 2000:       | 387 ( 6.58% )        |
| 2001:       | 708 (12.03% )        |
| 2002:       | 1268 (21.55% )       |
| 2003:       | 2907 (49.40% )       |
| cpan:       | 5885 (100.00% )      |

# Applications

- The One Liner
- `mod_perl`
- Templates
- Spam Assassin
- LWP

# The One Liner

- Perl can also be invoked from the command line
- `perl -e 'print time' # accept prog on cmd line`
- `perl -p -e 's/use/utilize/'`
  - Assume `while (<STDIN>) {.... print }` around input
- `perl -i -p -e 's/use/utilize/' FILESPEC`
  - Edit files in place

Real programmers can write assembly  
code in any language.

-Larry Wall

# mod\_perl

- Integrates perl with Apache
  - Keep CGI resident in memory for faster startup
  - Custom handlers allow you to handle dynamic content easily, and at any stage of the request
  - slashdot.org uses mod\_perl to fetch templates from a database

# Template Engines

- So you want to separate HTML and logic?
  - Templates are the answer
  - TT2, Mason, HTML::Template
- Templates can read program data and call subs, but use a simpler language

```
[% IF session.username %]
Hello, [% session.username %]

[% ELSE %]
Registered users log in:
[% END %]
```

# Spam Assassin

- Perl based spam classification tool
- Heuristics, Bayesian filters, external lookups
- Daemon mode or procmail filter
- Per user configurations
- *install Mail::SpamAssassin* from CPAN

# LWP

- Series of modules to handle web requests
- GET/POST/HEAD requests
- Robot classes to automatically respect robots.txt
- Multithreaded versions

# For Further Reading

- *Programming Perl*, 3ed (ORA)
- *Perl, The Complete Reference* (SAMS)
- *LWP and Perl* (ORA)
- <http://perl.com>
- <http://cpan.org>
- <http://winnipeg.pm.org>

# Q & A

If you guys want me to  
stop talking, you'd better  
ask some questions.

- Larry Wall

# Thanks!