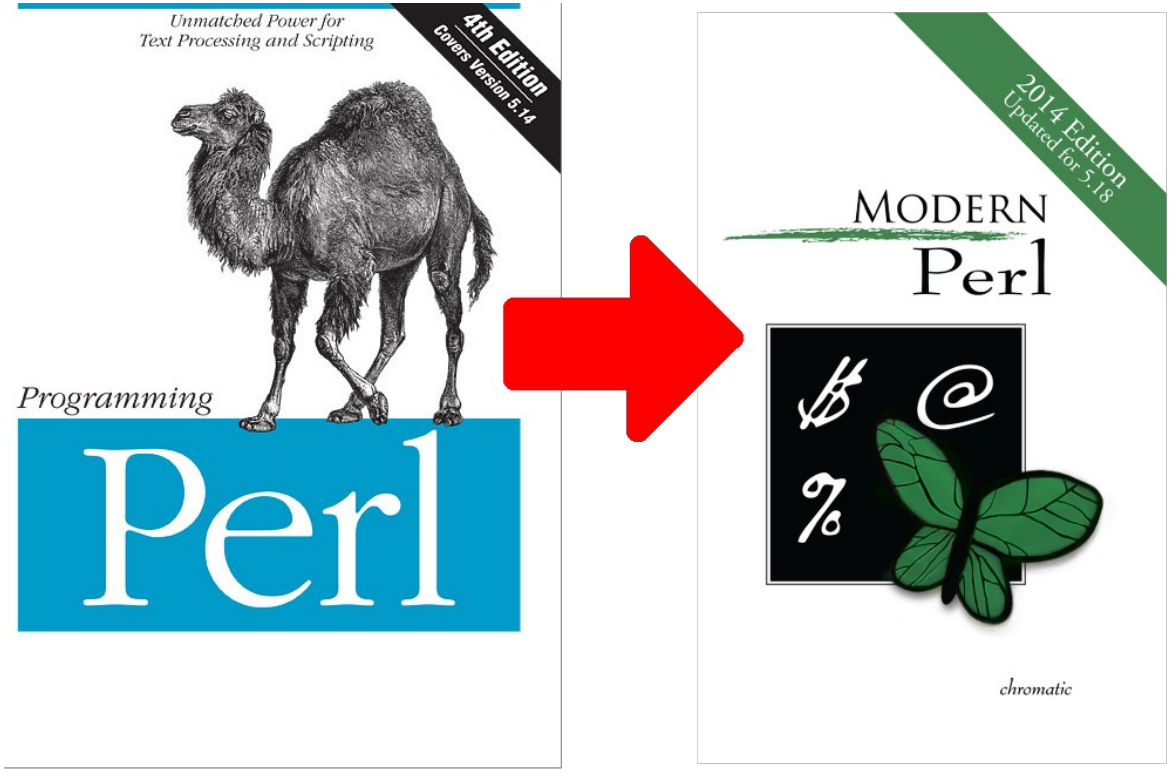
Thoroughly Modern Perl



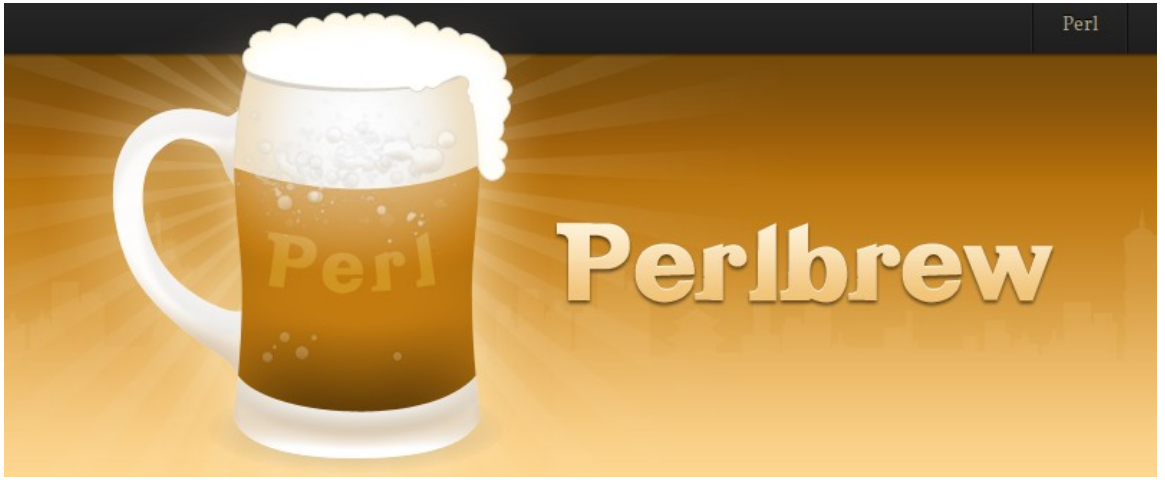
William Lindley
The Lindley Company LLC

Perl 5, Perl 6: Different animals

- Perl 6, announced in 2000, is still not a stable platform − actually a separate language
- Perl 5.20 released May 2014
- Perl 5 continues development stronger than ever

Perlbrew

- http://perlbrew.pl/
- Simple install:
  - \curl -L http://install.perlbrew.pl | bash
    (or)
  - sudo cpan App::perlbrew
  - perlbrew init



Perlbrew

- Manage multiple perl installations under your $HOME directory.
- Individual users or web applications operate independently
- Self-compiled (once!) to fit your system perfectly
- Don't need sudo to install modules (Pro: Can keep multiple versions across system for compatibility; Con: Each user's copy of perlbrew and modules must be updated separately when required)

Perlbrew is simple

- To install the latest stable release, and use it permanently:
  - perlbrew install perl-5.20.0
  - perlbrew switch perl-5.20.0
- All this requires only one extra line appended to ~/.bash_profile :
  - source ~/perl5/perlbrew/etc/bashrc

Baked In ~~Unicorns~~ Unicode

- Supports UTF-8 and other stream types in addition to ASCII
- UTF-8 is self-synchronizing: can find code point boundaries wihout reading from the beginning of the string.

Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Unicode in Perl

- **use utf8;** # at start of file
  tells Perl the *source code* is in Perl.
- **use 5.012;** *(or)* **use v5.14;** # or higher
  tells Perl to process strings internally as Unicode logically-wide characters
- **open FH, ">:utf8", "file";**
  opens a file (stream) with Unicode encoding layer
- **binmode(STDOUT, ":utf8");**
  reopens standard output with Unicode encoding

Unicode characters

- As a Perl one-liner

```
$ perl -e 'use v5.14; use charnames
":full"; binmode(STDOUT,":utf8");
print "\N{GREEK SMALL LETTER SIGMA}"'
σ
$
```

- http://www.unicode.org/charts/charindex.html

Seamless Unicode

```
use v5.14;
use utf8;

binmode(STDOUT,":utf8");
use Unicode::Collate;
my @words=qw(crow cräme cream crême);

print join(' ', Unicode::Collate->
    new->sort(@words));
```

→ cräme cream crême crow

Perl 5's Built-in Object System

- Added 1994, with Perl 4 → 5 transition
- A class is just a package (namespace).
- A method is just a subroutine
- Attributes are… stored however the class/module wants
- Instances are “blessed” references.
  - Package MyClass 1.07 {
    sub new {
      my $class = shift;
      return bless { }, $class; # { } here is an anonymous hash
    }
  }

Modern Perl: use Moose;

- Simple to code
  - package Cat {
    use Moose;
    }
- Automatically gives you functions:
  - my $muffin = Cat->new;

Moose: Methods

```
package Cat {
    use Moose;
    sub meow {
        my $self = shift;
        say 'Meow!';
    }
}
```

- my $muffin = Cat->new;
  $muffin->meow;

Moose: Attributes

```
package Cat {
    use Moose;
    has 'name', is => 'ro', isa => 'Str';
}

my $muffin=Cat->new( name => 'muffin' );
print $muffin->name;
```

Why this works: Syntactic Sugar

- Moose's documentation uses parentheses to separate attribute names and characteristics:

```
has 'name' => ( is => 'ro', isa => 'Str' );
```

- This is equivalent to:

```
has( 'name', 'is', 'ro', 'isa', 'Str' );
```

Integrating with Databases

- Brute force

```
my $sth = $dbh->prepare("INSERT INTO people
                (address, city, name, phone, state)
                VALUES (?, ?, ?, ?, ?)");
$sth->execute('42 Sister Lane', 'St. Louis',
        'Jimbo Bobson', '123-456-7890',
        'Louisiana');
```

With use SQL::Abstract;

```
my $sql = SQL::Abstract->new;
my %data = (
    name => 'Jimbo Bobson',          phone => '123-456-7890',
    address => '42 Sister Lane',     city => 'St. Louis',
    state => 'Louisiana',
);
my($stmt, @bind) = $sql->insert('people', \%data);
```

Which would give you something like this:

```
$stmt = "INSERT INTO people (address, city, name, phone, state)
            VALUES (?, ?, ?, ?, ?)";
@bind = ('42 Sister Lane', 'St. Louis', 'Jimbo Bobson', '123-456-7890', 'Louisiana');
```

These are then used directly in your DBI code:

```
my $sth = $dbh->prepare($stmt);
$sth->execute(@bind);
```

With use DBIX::Class;

- Create a schema class called MyApp/Schema.pm:

```
package MyApp::Schema;
use base qw/DBIx::Class::Schema/;

__PACKAGE__->load_namespaces();

1;
```

•Create a result class to represent artists, who have many CDs…

- in MyApp/Schema/Result/Artist.pm:

```
package MyApp::Schema::Result::Artist;
use base qw/DBIx::Class::Core/;

__PACKAGE__->table('artist');
__PACKAGE__->add_columns(qw/ artistid name /);
__PACKAGE__->set_primary_key('artistid');
__PACKAGE__->has_many(cds =>
'MyApp::Schema::Result::CD', 'artistid');

1;
```

Then you can…

```
# Connect to your database.

use MyApp::Schema;

my $schema = MyApp::Schema->connect($dbi_dsn,$user, $pass, \
%dbi_params);


# Query for all artists and put them in an array,

# or retrieve them as a result set object.

# $schema->resultset returns a DBIx::Class::ResultSet

my @all_artists = $schema->resultset('Artist')->all;


# Output all artists names

foreach $artist (@all_artists) {

  print $artist->name, "\n";

}
```

```
# Create a result set to search for artists.

# This does not query the DB.

my $johns_rs = $schema->resultset('Artist')->search(

  # Build your WHERE using an SQL::Abstract structure:

  { name => { like => 'John%' } }

);


# Execute a joined query to get the cds.

my @all_john_cds = $johns_rs->search_related('cds')->all;
```

Evolution of Perl Webstuff

- Apache − perl.cgi
- Apache+mod_perl
- Nginx − starman − framework (mojolicious)

Mojo: Rapid development

- http://mojolicio.us/
- Think “Ruby on Rails” in the Perl world
- Based on Moose
- Has a “lite” version (like Sinatra) but can easily turn Mojolicious::Lite programs into full-blown systems
- RESTful routes, plugins, commands, Perl-ish templates (Template::Toolkit), session management, form validation, testing framework, static file server…

Mojo: Easy to create and deploy

- Auto-detects standalone, CGI, PSGI environment
- Contains a very portable non-blocking I/O HTTP and WebSocket server with Mojo::Server::Daemon. It is usually used during development and in the construction of more advanced web servers, but is solid and fast enough for small to mid sized applications.
- Supports TLS and WebSockets out of the box
- JSON, HTML/XML parser, CSS selector support

Three line web application

```
use Mojolicious::Lite;
get '/' => {text => 'Hello World!'};
app->start;
```

- Then run it with:

```
$ morbo hello.pl
Server available at http://127.0.0.1:3000.
$ curl http://127.0.0.1:3000/
Hello World!
```

Thoroughly Modern Perl

William Lindley
The Lindley Company LLC
http://wlindley.com
904-404-5512