If you have to debug with print()

Oliver Gorwits

TVPM - 19th June 2013

Debugging with print()

Debugging with print()

- Hey... everyone does it ;-)

Debugging with print()

- Hey... everyone does it ;-)
- Remember to use Standard Error

```
print STDERR “>>> entering sub()\n”;
```

- Plays nicely with prove
- Friendly to logging systems
- Or simply, warn . . .

Data::Dumper

- Ships with Perl since forever
- Copes with most sorts of data

```
use Data::Dumper;
print STDERR Dumper $data_structure_ref;

    $VAR1 = [
               {
                 '69' => 'c',
                 '1'  => 'a',
                 '19' => 'b'
               }
             ];
```

Data::Dumper

- Ships with Perl since forever
- Copes with most sorts of data

```
use Data::Dumper;
print STDERR Dumper $data_structure_ref;

    $VAR1 = [
              {
                '69' => 'c',
                '1'  => 'a',
                '19' => 'b'
              }
            ];
```

Sets Var

Parseable

Debug: Human Inspection

- Parseable Perl? No thanks.
  - We have Storable, YAML, etc. for this!
- Setting a variable? No thanks.
  - Rarely necessary
  - Others (e.g. Data::Dump) don't bother
- I'd like DTRT, pretty, and configurable

Data::Printer

- COLOUR!! OMG the future is NOW
- Pretty == Human Readable (not Perl)
- Defaults to Standard Error
- Simple interface
- Highly Configurable

```
use Data::Printer;    # or just "use DDP" for short

p @array;             # no need to pass references

[
    [0] "a",
    [1] "b",
    [2] undef,
    [3] "c",
]
```

```
use Data::Printer;    # or just "use DDP" for short

p @array;             # no need to pass references

[
    [0] "a",
    [1] "b",
    [2] undef,
    [3] "c",
]
```

```
my $obj = SomeClass->new;

p($obj); # inspect an object

\ SomeClass  {
    Parents       Moose::Object
    Linear @ISA   SomeClass, Moose::Object
    public methods (3) : bar, foo, meta
    private methods (0)
    internals: {
        _something => 42,
    }
}
```

Why We Likes It

- Sane defaults
- Easily customisable at Import
- Load settings from a `.dataprinter` file
- Add custom dumpers/filters to your classes
- Few dependencies (= fast to install)

Opt-In Logging Example

```
use Class::Load ();

# try to load Data::Printer for debug output
if (Class::Load::try_load_class('Data::Printer')) {

    # load DDP with options
    Data::Printer->import({ class => { expand => 'all' } });
}

# later, when it’s time to log at debug level...

if (Class::Load::is_class_loaded('Data::Printer')) {
    $self->logger->log('debug', Data::Printer::p($thing));
}
```

End

Bonus: *top DBIx::Class tip*

```
$> DBIC_TRACE=1 DBIC_TRACE_PROFILE=console my_app...
```

```
SELECT COUNT( * )
  FROM (
    SELECT dp.ip AS left_ip, d1.dns AS left_dns, dp.port AS left_port, dp.duplex AS left_duplex, di.ip AS right_ip, d2.dns
AS right_dns, dp.remote_port AS right_port, dp2.duplex AS right_duplex
    FROM (
      SELECT device_port.ip, device_port.remote_ip, device_port.port, device_port.duplex, device_port.remote_port
        FROM device_port
      WHERE device_port.remote_port IS NOT NULL AND device_port.up AND NOT ILIKE '%down%'
      GROUP BY device_port.ip, device_port.remote_ip, device_port.port, device_port.duplex, device_port.remote_port
      ORDER BY device_port.ip
     ) dp
    LEFT JOIN device_ip di
      ON dp.remote_ip = di.alias
    LEFT JOIN device d1
      ON dp.ip = d1.ip
    LEFT JOIN device d2
      ON di.ip = d2.ip
    LEFT JOIN device_port dp2
      ON di.ip = dp2.ip AND dp.remote_port = dp2.port
 WHERE di.ip IS NOT NULL AND dp.duplex <> dp2.duplex AND dp.ip <= di.ip AND dp2.up AND NOT ILIKE '%down%'
 ORDER BY dp.ip
) me
```