Perl Tricks

Stefan Hornburg (Racke)

Contents

OO in Perl

namespace::clean

It is a best practice to use `namespace::clean` in your class or your role after `use ...` statements. Otherwise imported functions double as methods for your class, which can have unexpected side effects which are really hard to track down.

Using Moo

Types for Moo

We can recommend to use Type::Tiny instead of MooX::Types::Mooselike. The latter is not Moose-friendly, while Type::Tiny types get inflated to full Moose types if you happen to play with them in Moose. Also they are faster and we get better coercion in Moo for free.

One thing you may not have spotted for Type::Tiny which is useful... it overloads | and & for use in isa - very tidy.

```
use Moo;
use Types::Standard qw/ArrayRef HashRef InstanceOf/;

has fields => (
    is => 'ro',
    isa => ArrayRef [ InstanceOf ['Template::Flute::Form::Field'] ],
);
```

ArrayRef

```
has config_files => (
    is      => 'ro',
    isa     => ArrayRef,
    default => sub { [] },
);
```

Maybe

Sometimes it makes sense to allow `undef` as a value, e.g. if the builder isn't able to produce the promised type:

```
has config => (
    is      => 'ro',
    isa     => Maybe[HashRef],
    lazy    => 1,
    builder => '_build_config',
);
```

Speed ups

- Class::XSAccessor
- Type::Tiny::XS

UTF-8

Ensure that me use UTF-8 encoding for the standard input/output streams:

```
use open ':std', ':encoding(utf-8)';
```

Using DateTime

Converting JSON dates

APIs often using JSON as output. A JSON date looks like that:

Formatting

```
DateTime->now->strftime('%Y-%m-%d');
```

Week

This function determines the first day of the week for a given year:

```
sub first_day_of_week
{
  my ($year, $week) = @_;

  # Week 1 is defined as the one containing January 4:
  DateTime
    ->new( year => $year, month => 1, day => 4 )
    ->add( weeks => ($week - 1) )
    ->truncate( to => 'week' );
}
```

Parsing

Text

Split text into an array of lines:

```
my @lines = split /\r?\n/, $content;
my %email_headers;

for my $line (@lines) {
  if ($line =~ /^(From|To):\s(.*)/) {
    $email_headers{lc($1)} = $2;
  }
}
```

List of arguments

The simplest way to parse a list of arguments separated by whitespace is the *split* function:

```
my $arglist = 'foo bar baz';

my @args = split (/\s+/, $arglist));
```

This doesn't work with a list of files when it includes filenames with whitespaces:

- 'foo bar'
- 'baz'

You can use the *shellwords* function from the Text::ParseWords module in this case:

```
use Text::ParseWords;
my $arglist = '"foo bar" baz';

my @args = shellwords($arglist);
```

Regular expresssions

Turn blank seperated text into a CSV

```
perl -pe 's/\p{Blank}+/;/g' dns2.txt > dns.csv
```

Wrap all urls in a text into a HTML link

```
$text =~s#(https?://\S+?)(\.*[\s\<,])#<a href="$1">$1</a>$2#igs;
```

Recipes

Print version of installed Perl modules

There is a very useful module called *V* for printing out version information about installed Perl modules:

```
perl -MV=Dancer2

Dancer2
    /home/racke/perl5/perlbrew/perls/perl-5.24.0/lib/site_perl/5.24.1/Dancer2.pm: 0.205000
    /home/racke/perl5/perlbrew/perls/perl-5.24.0/lib/site_perl/5.24.0/Dancer2.pm: 0.204001
```

Convert images with Image::Magick

Resize by percents

```
$ convert mypic.jpg -resize 25% mypicsmall.jpg
```

Create transparent picture

```
$ convert -size 1000x1000 xc:transparent any-square.png
```

CPAN modules - the good and the bad

`File::Slurp` is deprecated, use for example `Path::Tiny` instead.

Charts

Chart::Clicker

Writing tests

Test for warnings

Always use Test::Warnings to capture unexpected warnings in your tests:

```
use Test::Warnings;
```

Test whether result is one of multiple values

```
use Test::Deep;

my $status = $mws->GetServiceStatus;

cmp_deeply $status, any(qw/GREEN GREEN_I YELLOW RED/),
   "Test response of GetServiceStatus API method";
```

Skipping tests

Often tests are using an optional module or external resources. In that case we just skip these tests if we haven't all pieces in the right place.

For example, if an environment variable isn't set properly:

```
if ($ENV{SOLR_URL}) {
    $solr_url = $ENV{SOLR_URL};
}
else {
    plan skip_all => "Please set environment variable SOLR_URL.";
}
```

Provide useful information with `diag`

Even if you think your tests won't fail, they might. In this case it is very helpful to provide additional information to trace down the source of the problem:

```
# check output

my @matches = $out =~ /Blue ball/g;

ok (@matches == 2, 'Test replacement in both lists')
    || diag "Matches: ", scalar(@matches), "Output: $out";
```

A good alternative is to use `is` instead of `ok`, which automatically provides diagnostics:

```
# check output

my @matches = $out =~ /Blue ball/g;

is (scalar(@matches), 2, 'Test replacement in both lists');
```

Also other test functions like `isa_ok` print further information.

Test coverage

Devel::Cover makes test coverage easy in your distribution:

```
cpanm Devel::Cover
perl Makefile.PL
make
cover -test
```

You can find more information in the blog post from Neil Bower.

Installing CPAN modules

cpanm

You can install Perl modules from Git repositories, e.g.

```
cpanm git@github.com:interchange/Amazon-MWS.git@topic/amazon-pay-2020-01-28
```

The part after the @ is a branch, tag or hashsum.

Debian prerequisites

Here we show you which libraries you need to install for binary Perl modules (Debian and Ubuntu).

DBD::MySQL

This works for MySQL and MariaDB:

```
$ apt install libmysqlclient-dev
```

DBD::ODBC

```
$ apt install unixodbc-dev
```

DBD::Pg

```
$ apt install libpq-dev
```

File::LibMagic

```
$ apt install libmagic-dev
```

Imager

You need to install a number of development packages for the different image types *before* installing Imager, e.g. on Debian:

```
$ apt install libjpeg-dev libpng-dev libgif-dev libtiff-dev libfreetype6-dev
```

Net::LibIDN2

```
$ apt install libidn2-dev
```

Net::SSLeay, Crypt::OpenSSL::Random, Crypt::OpenSSL::X509, ...

```
$ apt install libssl-dev
```

IO::Socket::SSL

```
$ apt install zlib1g-dev
```

XML::Parser, XML::Twig, ...

```
$ apt install libexpat1-dev
```

XML::LibXML

```
$ apt install libxml2-dev
```

Tips for other modules

Dancer2

For more speed, install recommended modules as well:

```
cpanm --installdeps --with-recommends Dancer2
```

These are XS modules replacing pure Perl modules, e.g. *HTTP::XSHeaders* or *Type::Tiny::XS*.

Patching CPAN modules

If you want to make a patch for a CPAN module which doesn't provide a Git repository, Git::CPAN::Patch comes in very handy:

```
% git-cpan clone WebService::Xero
creating WebService-Xero
created tag 'v0.10' (7dbe0fdd5c54307ce6ea6d6943ddf529e1a7ab8c)
created tag 'v0.11' (082ce463c0dec679710d0eecfbbaf47262526bff)
```

Create a branch for your patch:

```
% git checkout -b topic/enable-put-method
```

After done with patching, you can submit the changes as follows:

1. add your changes and commit
2. push patch to CPAN

For example:

```
% git add -u
% git commit -m "Enable PUT method in Xero agent."
[topic/enable-put-method 76c60c1] Enable PUT method in Xero agent.
%
```

Writing CPAN modules

Resources

Please add resources for meta::cpan to your Makefile.PL, e.g.

```
META_MERGE => {
    resources => {
        repository  => 'https://github.com/interchange/interchange6-schema.git',
        bugtracker => 'https://github.com/interchange/interchange6-schema/issues',
        IRC => 'irc://irc.freenode.net/#interchange',
    },
},
```

Prerequisites and minimum versions

Test::Deep

0.114 if you use `noneof`

Type::Tiny

0.008 if you use `InstanceOf`

Travis

Before you commit and push your `.travis.yml` file, make sure that file is valid:

```
travis lint .travis.yml
```

Perl versions

The lastest Perl version supported by Travis is `5.30` as of September 2020.

Coverage reports

```
language: perl
perl:
  - "5.10"
  - "5.12"
  - "5.14"
  - "5.16"
  - "5.18"
  - "5.20"
```

```
  - "5.22"
  - "5.24"
  - "5.28"
  - "dev"     # installs latest developer release of perl
  - "blead"   # builds perl from git
matrix:
  include:
    - perl: 5.28
      env: COVERAGE=1   # enables coverage+coveralls reporting
  allow_failures:
    - perl: blead       # ignore failures for blead perl
sudo: false             # faster builds as long as you don't need sudo access
before_install:
  - eval $(curl https://travis-perl.github.io/init) --auto
```

Please see Travis helpers for complete reference.

Scripts

Add missing columns with default values to CSV file:

```
$ perl -i -pe 's/$/;0;/' data.csv
```

This one-liner removes carriage returns in a file:

```
$ perl -i -pe 's/\r//g' data.csv
```

Turn file with multiples lines into a comma separated list:

```
$ cat myfile.txt
FOO
BAR
BAZ

$ perl -pe 's/\r?\n/,/g' myfile.txt
FOO,BAR,BAZ
```

Linuxia Wiki

Stefan Hornburg (Racke)

Perl Tricks

wiki.linuxia.de