



Fun With Perl

5.10.1

Mike Stok <mike@stok.ca>
Toronto Perl Mongers, 28 Jan 2010

Thursday, February 11, 2010

Version really doesn't matter for this project, didn't use "say" or "//", don't care about performance.

Slides 11 & 12 updated after useful input from mirod at xmltwig.com (11 Feb 2010).



Building to.pm.org

- Don't want to:
 - Change look or feel.
 - Write a content management system.
 - use Catalyst;
 - Feel like I'm at work.

Thursday, February 11, 2010

Some history about how we used to add meetings, and the desire to clean up the HTML, but not to get into the whole web 2.0 quagmire. Simple, small, fun.



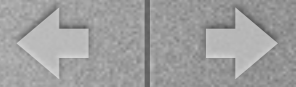
Building to.pm.org

- Do want to:
 - Improve the HTML.
 - Start afresh.
 - Try some modules.



local::lib

- Ubuntu has 5.10.0, I built 5.10.1 as root and installed it in `/usr/local/mcnugget/...`
- Download local-lib from CPAN and install as myself
 - `perl Makefile.PL --bootstrap`
 - `make test && make install`
 - `echo 'eval $(perl -I$HOME/perl5/lib/perl5 -Mlocal::lib)' >> ~/.bashrc`



Bondage & Discipline

- Use Ubuntu's packaged tools
 - `perltidy -pbp`
 - `libperl-critic-perl`
 - ```
mike@mcnugget:~/work/tpm$ make critic
perlritic --brutal code/build-index code/TPM/*.pm
code/build-index source OK
code/TPM/Meeting.pm source OK
```
- W3C HTML validation service

Thursday, February 11, 2010

I don't particularly like `perltidy -pbp` layout, but it's consistent and I don't feel strongly enough to customize it. `-pbp` in `~/.perltidycrc`, all examples are formatted this way.

Same for `perlritic`, but I want to ditch the RCS keywords and versions for now, so a small `perlcritirc`.



# “Perl::Critic”

```
Set up the attribute accessors at compile time

BEGIN {
 my @ATTRIBUTES = qw/ title venue timestamp date content /;

 for my $attr (@ATTRIBUTES) {
 ## no critic 'TestingAndDebugging::ProhibitNoStrict'
 no strict 'refs';
 *{ __PACKAGE__ . q{::} . $attr } = sub {
 my $self = shift;
 $self->_loaded_or_croak;
 return $self->{"_ $attr"};
 };
 }
}
```

Thursday, February 11, 2010

In the TPM::Meeting module we can do “dangerous stuff” as perlcritic lets us turn off specific warnings for small (!) scopes. (c.f. lint)



# Interesting Modules Used

- `use Path::Class;`  
`use File::Basename;`  
`use Readonly;`  
`use English '-no_match_vars';`  
`use Template;`  
`use Carp;`  
`use File::Find::Rule;`  
`use POSIX 'strftime';`  
`use TPM::Meeting;`
- `use XML::Twig;`  
`use Date::Parse;`

Thursday, February 11, 2010

The code to write out the index page and the TPM::Module use several modules. Most of these are familiar, we'll just look at the red ones.



# Path::Class

```
in:
directory under which sections live in separate html files
out:
sorted list of sections { name => ..., content => ... }
#
File names determine the section names, the files are assumed to
contain valid HTML.
sub get_sections {
 my ($sections_dir) = @_ ;
 Readonly my $SECTION_SUFFIX => '.html' ;

 return [
 map {
 { name => basename($_, $SECTION_SUFFIX),
 content => scalar file($_)->slurp,
 }
 }
 sort
 grep {/\Q$SECTION_SUFFIX\E/z/smox} dir($sections_dir)->children
];
}
```

Thursday, February 11, 2010

“Functional” style code with data going “east”. Perltidy doesn’t really make it that clear. Can use comments to force line breaks. Path::Class makes things a little easier than my usual directory reading code, gluing paths together, reading files. Very convenient.



# Template

```
<!-- each year's meeting links -->
[% FOR year IN years_meetings %]
 [% "" IF loop.first %]

 [% FOR meeting IN year.meetings %]
 [% meeting.date | html %][% ", " IF NOT loop.last %]
 [% END %]

 [% "" IF loop.last %]
[% END %]
```

Thursday, February 11, 2010

An oldie, but I had forgotten about `loop.first` and `loop.last`. Way too much in Template Toolkit but this is the only (old but) new to me trick.

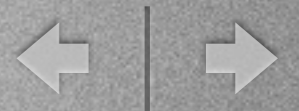


# XML::Twig (0)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<meeting>
 <details>
 <title>Modules and Tips</title>
 <datetime>2010-01-28T18:45</datetime>
 <venue>Nexient Learning; Room 15 on the 8th floor</venue>
 </details>
 <description>
 <p class="synopsis">A series of quick talks covering various modules which
 people have found useful or instructive. Some are old standbys, some are
 new.</p>
 <!-- etc. -->
 </description>
</meeting>
```

Thursday, February 11, 2010

As the meeting bits have XHTML fragments, might as well store it with the metadata in an XML file and process that. Poorly formed XHTML will not parse – Simple structure (already changing to accommodate multiple speakers...)



# XML::Twig (I)

```
my $t = XML::Twig->new(
 twig_roots => {
 'meeting/details/title' =>
 sub { $self->stash_text(@_, 'title'); },
 'meeting/details/venue' =>
 sub { $self->stash_text(@_, 'venue'); },
 'meeting/details/datetime' => sub { $self->stash_datetime(@_); },
 'meeting/description' =>
 sub { $self->stash_xhtml(@_, 'content'); },
 },
 pretty_print => 'indented',
);
$t->safe_parsefile($file_name)
 or croak "failed to parse and process $file_name ($EVAL_ERROR)";
```

Thursday, February 11, 2010

XML::Twig is yet another XML module, seems like fun and I haven't scratched the surface. Look at the handlers next...



# XML::Twig (2)

```
sub _stash_text {
 my ($self, $twig, $elt, $attr) = @_;
 $self->{"_ $attr"} = $elt->text;
 return;
}

sub _stash_datetime {
 my ($self, $twig, $elt) = @_;
 my $text = $elt->text;
 my $epoch_secs = str2time($text);
 defined $epoch_secs or croak "failed to parse date time '$text'";
 $self->{_timestamp} = $epoch_secs;
 $self->{_date} = strftime('%a %e %b %Y %R %Z', localtime $epoch_secs);
 return;
}

sub _stash_xhtml {
 my ($self, $twig, $elt, $attr) = @_;
 $self->{"_ $attr"} = $elt->inner_xml;
 return;
}
```

Thursday, February 11, 2010

Last one is peeling out the XHTML, so I wanted to strip the <description> and </description> tags and save the stuff between with all its mark-up. If I have badly formed XHTML then the file will not parse.



# File::Find::Path

```
in:
directory for root of tree
out:
ref to list of TPM::Meeting objects
sub get_meetings {
 my ($meetings_dir) = @_;
 my @meetings;

 for my $file (find(file => name => '*.xml', in => $meetings_dir)) {
 my $meeting = TPM::Meeting->new;
 $meeting->load_file($file);
 push @meetings, $meeting;
 }

 return \@meetings;
}
```

Thursday, February 11, 2010

Not sure how I was going to structure the XML directories, so this is pretty simple to read and use.



# Modules Not Used

- High hopes for:
  - **Modern::Perl** (... not for this project.)
- Should maybe have used:
  - **Module::Starter**
  - **Class::Exception**