Perl 201

Jan Pazdziora
Principal Software Engineer
Satellite Engineering, Red Hat

4th November 2011

Hello world

```
#!/usr/bin/perl
use strict;
use warnings FATAL => 'all';
use utf8;

binmode STDOUT, ':utf8';
print "¡Hola, mundo!\n";
1;

=head1 NAME

Hello - greetings to the world.

=head1 SYNOPSIS

    hello
```

Parsing email (MIME::Parser)

```
use strict;
use warnings FATAL => 'all';

use MIME::Parser ();
my $entity = new MIME::Parser->parse(\*STDIN);
my $head = $entity->head;
print "F: ", $head->get('From'), "\n";
print "S: ", $head->get('Subject'), "\n";
if (not $entity->effective_type =~ m!^text/!) {
    for my $p ($entity->parts_DFS) {
        if ($p->effective_type =~ m!^text/!) {
            $entity = $p;
            last;
        }
    }
}
print $entity->bodyhandle->as_string;
```

Parsing email (MIME::Parser) (cont'd)

```
use MIME::Parser ();
use Mail::Address ();
use HTML::FormatText ();
binmode STDOUT, ':utf8';

my $parser = new MIME::Parser();
$parser->output_to_core(1);
$parser->use_inner_files(1);
my $entity = $parser->parse(\*STDIN);

my $head = $entity->head;
my $from = $head->get('From');
for (eval { (Mail::Address->parse($from))[0]->address }) {
    $from = $_;
}
my $subject = $head->get('Subject');
for ($from, $subject) {
    $_ = MIME::WordDecoder::mime_to_perl_string($_);
    chomp;
}
```

Parsing email (MIME::Parser) (cont'd)

```
print "F: $from\nS: $subject\n";
```

Parsing email (MIME::Parser) (cont'd)

```
if (not $entity->effective_type =~ m!^text/!) {
    for my $p ($entity->parts_DFS) {
        if ($p->effective_type =~ m!^text/!) {
            $entity = $p;
            last;
        }
    }
}

my $body = $entity->bodyhandle->as_string;
for my $charset ($entity->head->mime_attr('content-type.charset')) {
    $body = Encode::decode($charset, $body);
}
if ($entity->effective_type =~ m!^text/html!) {
    $body = HTML::FormatText->format_string($body, leftmargin => 0);
}

$body =~ s/(\s)\s+/$1/g;
$body =~ s/^\s+//;
print $body;
```

Parsing email (Mail::Message)

```
use Mail::Message ();
use HTML::FormatText ();
binmode STDOUT, ':utf8';

my @lines = <STDIN>;
shift @lines if @lines and $lines[0] =~ /^From /;
my $message = Mail::Message->read(\@lines);
my $from = ($message->from)[0]->address;
my $subject = $message->study('Subject');
print "F: $from\nS: $subject\n";

($message) = ( (grep { $_->contentType =~ m!^text/! }
        $message->parts('RECURSE')), $message);
my $body = $message->decoded(charset => 'PERL');
if ($message->contentType =~ m!^text/html!) {
    $body = HTML::FormatText->format_string($body);
}
$body =~ s/(\s)\s+/$1/g;
$body =~ s/^\s+//;
print $body;
```

Parsing email (Email::MIME)

```
use Email::MIME ();
use HTML::Strip ();
use HTML::Entities ();
binmode STDOUT, ':utf8';

my $content = join '', <STDIN>;
$content =~ s/^From .*\n//;
my $message = new Email::MIME($content);
my $from = (Mail::Address->parse($message->header('From')))[0]->address;
my $subject = $message->header('Subject');
print "F: $from\nS: $subject\n";
```

Parsing email (Email::MIME) (cont'd)

```
my $body;
$message->walk_parts(
    sub {
        return if defined $body;
        my $part = shift;
        if ($part->content_type =~ /^text\/html/i) {
            $body = $part->body_str;
            $body = HTML::Entities::encode_entities($body, "\200-\777");
            $body = HTML::Strip->new->parse($body);
        } elsif ($part->content_type =~ /^text\//i) {
            $body = $part->body_str;
        }
    }
);

$body =~ s/\r/\n/g;
$body =~ s/(\s)\s+/$1/g;
$body =~ s/^\s+//;
print $body;
```

Parsing XML

```
use XML::LibXML ();
use XML::LibXML::XPathContext ();
binmode STDOUT, ':utf8';

my $file = '/etc/gconf/gconf.xml.defaults/%gconf-tree-cs.xml';
my $dom = XML::LibXML->load_xml(location => $file);
my $xc = new XML::LibXML::XPathContext($dom);
for my $node ( $xc->findnodes('
    /gconf
        /dir[@name="schemas"]
        /dir[@name="system"]
        /dir[@name="proxy"]
        /entry[@name="ftp_host"]
        /local_schema
        /longdesc
        /text()
        ')
    )) {
    print $node->parentNode->parentNode->getAttribute('short_desc'), "\n";
    print $node->nodeValue, "\n";
```

Parsing XML (cont'd)

```
}
```

Work with XML and digests

```
use XML::LibXML ();
use XML::LibXML::XPathContext ();
use IO::File ();
use Digest ();

my $repomd = shift;
$repomd .= "/repomd.xml" if -d $repomd;

my $dom = XML::LibXML->load_xml(location => $repomd);
my $xc = new XML::LibXML::XPathContext($dom);
$xc->registerNs('repo', 'http://linux.duke.edu/metadata/repo');

my ($comps) = $xc->findnodes('/repo:repomd/repo:data[@type="group_gz"]');
my $exp_checksum = $comps->getChildrenByTagName('checksum')
                                                    ->string_value;

my ($exp_type) = map { $_->getAttribute('type') }
                                $comps->getChildrenByTagName('checksum');
$exp_type =~ s/^sha/SHA-/;
my $d = new Digest($exp_type) or die "Unknown checksum $exp_type\n";
```

Work with XML and digests (cont'd)

```
my ($comps_file);
for ($comps->getChildrenByTagName('location')) {
    $comps_file = $_->getAttribute('href');
}
$comps_file =~ s!^.*/!!;
my ($full_comps) = ($repomd =~ /(^.*\/)/);
$full_comps .= $comps_file;

my $fh = new IO::File $full_comps, 'r'
                              or die "Error reading $full_comps: $!\n";
while (<$fh>) {
    $d->add($_);
}
my $hex = $d->hexdigest;
if ($hex eq $exp_checksum) {
    print "Checksum OK\n";
} else {
    print "Expected $exp_checksum\n     Got $hex\n";
}
```

Interact with the Web

```
$ perl -MLWP::Simple \
   -e 'getprint "http://search.cpan.org/search?query=LWP"' \
   | grep h2 | head -5
  <p><h2 class=sr><a href="/~gaas/libwww-perl-6.03/lib/LWP.pm"><b>LWP</b></a
  <p><h2 class=sr><a href="/~mlehmann/Coro-6.06/Coro/LWP.pm"><b>Coro::LWP</b
  <p><h2 class=sr><a href="/~mmorgan/LWP-UserAgent-Mockable-1.10/lib/LWP/Use
  <p><h2 class=sr><a href="/~gaas/libwww-perl-6.03/lib/LWP/Authen/Ntlm.pm"><
  <p><h2 class=sr><a href="/~autrijus/LWP-Authen-Wsse-0.05/lib/LWP/Authen/Ws
```

```
$ perl -MLWP::Simple \
   -e 'getprint "https://bugzilla.redhat.com/buglist.cgi?bug_status=NEW&comp
```

```
$ GET -H 'User-Agent: Mozilla/5.0' -Se 'http://www.google.com/'
```

Interact with the Web (cont'd)

```
use LWP;
my $agent = new LWP::UserAgent(agent => 'Mozilla/5.0', timeout => 10);
my $request = new HTTP::Request(
    GET => 'http://www.google.com/',
    new HTTP::Headers(
        'Accept-Language' => 'cs',
        'Accept' => 'text/html,application/xhtml+xml',
    ));
my $response = $agent->request($request);
if ($response->is_success) {
    print $response->header('Content-Type'), "\n";
    print $response->content;
} else {
    print STDERR $response->error_as_HTML;
}
```

Longer sessions on the Web

```
use WWW::Mechanize;
my $m = new WWW::Mechanize(agent => 'Mozilla/5.0');
$m->add_header('Accept-Language' => 'nl');

$m->get('http://www.google.com/');
$m->follow_link(text_regex => qr/geavanceerd.+zoeken/i);
print $m->uri, "\n";

$m->submit_form( fields => { as_q => 'www mechanize', });

if ($m->find_link(text_regex => qr/cpan\.org/i)) {
    $m->follow_link(text_regex => qr/cpan\.org/i);
    print $m->uri, "\n";

    $m->follow_link(text_regex => qr/download/i);
    my $filename = $m->uri;
    $filename =~ s!^.*/!!;
    print "Got [$filename] length [@{[ length($m->content) ]}]\n";
}
```

XMLRPC

```
use XMLRPC::Lite ();

my $client = XMLRPC::Lite->proxy('http://xmlrpc.rhn.redhat.com/rpc/api');
my $response = $client->call('api.getVersion');
if (not $response->fault) {
    print $response->result, "\n";
}
```

Databases

```
use DBI;
my $dbh = DBI->connect('dbi:Pg:dbname=perl201',
    'perl201user', 'teslo',
    { AutoCommit => 0, RaiseError => 1 });
my $sth = $dbh->prepare('insert into x201 values (3, ?)');
$sth->execute('horse');
```

```
$ ./dbi
Issuing rollback() due to DESTROY without explicit disconnect() of DBD::Pg::
```

```
$dbh->commit;
$dbh->disconnect;
```

```
$dbh->{pg_enable_utf8} = 1;

my $sth = $dbh->prepare('select id, name from x201 order by id');
$sth->execute;
while (my $data = $sth->fetchrow_hashref) {
    use Data::Dumper; print Dumper $data;
}
```

Databases (cont'd)

```
$dbh->{pg_enable_utf8} = 1;
$dbh->{FetchHashKeyName} = 'NAME_uc';

my $data = $dbh->selectall_arrayref('
    select id, name, upper(name) x
    from x201
    where id = ?
    order by name',
    { Slice => {} }, $ARGV[0]);
```

```
$data = $dbh->selectrow_hashref('
        select name, upper(name) x
        from x201
        where id = ?
        ', {}, $ARGV[0]);
```

Self-assessment

- Given remote URL of Fedora repository and local cache directory, populate the directory with the same files yum would. Verify checksums and do not re-download data that is already available locally. On the other hand, purge and re-retrieve files that got updated on the remote site.
- Add a comment to

```
https://bugzilla.redhat.com/show_bug.cgi?id=1
```

  Reopen it if it is currently closed, and mark it as closed if it is open. Do it with Perl script.
- Write a script to store uids, logins, and passwords of local users into database table. It should not only handle the initial loading of data but also updates upon subsequent runs.

Final questions?

No?

OKay. Thank you!