Perl 301

Jan Pazdziora
Principal Software Engineer
Satellite Engineering, Red Hat

16th December 2011

Mirroring yum metadata

```
use LWP::UserAgent ();
use XML::LibXML ();
use Digest ();
use IO::File ();
```

```
my ($url, $dir) = @ARGV;
if (not defined $dir) { die "Usage: $0 yum-remote-url local-directory\n"; }
if (-e $dir and not -d $dir) {
    die "The target [$dir] exists but is not directory.\n";
}
if (not -d $dir) {
    my @segments = ($dir =~ m!(.[^/]*)!g);
    my $path = '';
    while (@segments) {
        $path .= shift(@segments);
        if (not -d $path) {
            print "mkdir $path\n";
            mkdir $path or die "Error creating [$path]: $!\n";
        }
    }
}
```

Mirroring yum metadata (cont'd)

```
sub check_checksum {
    my ($file, $digest) = @_;
    my $fh = new IO::File $file, 'r' or die "Error reading $file: $!\n";
    while (<$fh>) {
        $digest->add($_);
    }
    $fh->close();
    return $digest->hexdigest;
}
my %extra_files;
@extra_files{ glob("$dir/*") } = ();
my $ua = new LWP::UserAgent();
if (not $url =~ /\/repodata\/repomd\.xml$/) {
    $url .= '/repodata/repomd.xml';
}
my $repomd = $ua->mirror($url, "$dir/repomd.xml");
delete $extra_files{"$dir/repomd.xml"};
my $dom = XML::LibXML->load_xml(location => "$dir/repomd.xml");
my $xc = new XML::LibXML::XPathContext($dom);
$xc->registerNs('repo', 'http://linux.duke.edu/metadata/repo');
```

Mirroring yum metadata (cont'd)

```
my $seen_error = 0;
for my $data ($xc->findnodes('/repo:repomd/repo:data')) {
    my $exp_checksum = $data->getChildrenByTagName('checksum')
                                                    ->string_value;
    my ($exp_type) = map { $_->getAttribute('type') }
                                $data->getChildrenByTagName('checksum');
    $exp_type =~ s/^sha/SHA-/;
    my $d = new Digest($exp_type) or die "Unknown checksum $exp_type\n";
    my ($file) = map { $_->getAttribute('href') }
                                $data->getChildrenByTagName('location');
    $file =~ s!^.*/!!;
    my $target_file = "$dir/$file";
    delete $extra_files{$target_file};
    if (-f $target_file) {
        my $checksum = check_checksum($target_file, $d);
        if ($checksum eq $exp_checksum) {
            # print "File [$file] already exists and correct checksum, not r
            next;
        }
        unlink $target_file;
```

Mirroring yum metadata (cont'd)

```
    }
    print "Will download [$file] to [$target_file]\n";
    $url =~ s![^/]+$!$file!;
    my $response = $ua->mirror($url, $target_file);
    if (not -f $target_file) {
        print "Failed to retrieve [$url]: @{[ $response->message ]}\n";
        $seen_error++;
        next;
    }
    $d->reset();
    my $hex = check_checksum($target_file, $d);
    if ($hex eq $exp_checksum) {
        print "\tChecksum OK\n";
    } else {
        print "\tBad checksum [$hex]\n";
        $seen_error++;
    }
}
```

Mirroring yum metadata (cont'd)

```
for (sort keys %extra_files) {
    print "Purging [$_]\n";
    unlink $_;
}
if ($seen_error) {
    exit 1;
}
```

Adding a comment to bugzilla

```
use WWW::Mechanize ();
use utf8;
my $mech = new WWW::Mechanize ();
$mech->get('http://bugzilla.redhat.com/show_bug.cgi?id=1');
$mech->quiet(1);
$mech->submit_form(with_fields => {
    'Bugzilla_login' => 'the-login',
    'Bugzilla_password' => 'hard-to-guess-password'
});
$mech->get('http://bugzilla.redhat.com/show_bug.cgi?id=1');
# $mech->dump_forms();
$mech->form_name('changeform');
my $status = $mech->value('bug_status');
$status = ($status eq 'CLOSED') ? 'ASSIGNED' : 'CLOSED';
$mech->submit_form(with_fields => {
    'bug_status' => $status,
    'comment' => join '', <DATA>,
});
__DATA__
Text komentáře.
```

Populating database table

```
use DBI ();
use utf8;

my %users;
open PASSWD, 'getent passwd |' or die "Error $!\n";
while (<PASSWD>) {
    my ($login, $pass, $uid) = split /:/, $_, 4;
    next if $uid < 500;
    $users{$login} = [ $uid ];
}
close PASSWD;

open SHADOW, 'getent shadow |' or die "Error $!\n";
while (<SHADOW>) {
    my ($login, $pass) = split /:/, $_, 3;
    next if not exists $users{$login};
    $users{$login}[1] = $pass;
}
close SHADOW;
```

Populating database table (cont'd)

```
my $dbh = DBI->connect('dbi:Pg:dbname=perl301', 'perl301user', 'teslo',
    { RaiseError => 1, AutoCommit => 0 });

my $sth = $dbh->prepare(q!
    select login, password, uid
    from users
    !);
$sth->execute;

my ($del_sth, $upd_sth);
while (my $row = $sth->fetchrow_hashref) {
    if (not exists $users{$row->{login}}) {
        $del_sth ||= $dbh->prepare(q!
            delete from users where login = ?
            !);
        $del_sth->execute($row->{login});
        print "Removing [$row->{login}]\n";
        next;
    }
```

Populating database table (cont'd)

```
    {
    no warnings 'uninitialized';
    if ($row->{uid} ne $users{$row->{login}}[0]
        or $row->{password} ne $users{$row->{login}}[1]) {
        print "Fixing [$row->{login}]\n";
        $upd_sth ||= $dbh->prepare(q!
            update users set uid = ?, password = ?
            where login = ?
            !);
        $upd_sth->execute($users{$row->{login}}[0],
            $users{$row->{login}}[1],
            $row->{login});
    }
    }
    delete $users{$row->{login}};
}
```

Populating database table (cont'd)

```
my ($ins_sth);
for my $login (keys %users) {
    $ins_sth ||= $dbh->prepare(q!
        insert into users (login, password, uid)
        values (?, ?, ?)
        !);
    $ins_sth->execute($login, $users{$login}[1], $users{$login}[0]);
    print "Adding [$login]\n";
}

$dbh->commit;

__END__

create table users (
    login varchar(64) not null primary key,
    password varchar(512),
    uid integer not null
);
```

Packages, modules

In `X.pm`, we have

```
package X;
sub x {
    print "This is x in ", __PACKAGE__, "\n";
}
1;
```

In our script, we call function in different package:

```
use X;
X::x();
```

```
$ perl s.pl
This is x in X
```

Classes, objects

In `Dir.pm`, we have

```
package Dir;
sub new {
    my $class = shift;
    my $self = { @_ };
    return bless $self, $class;
}
sub dir {
    my $self = shift;
    if (@_) {
        $self->{dir} = shift;
    }
    return $self->{dir};
}
sub parent {
    my $self = shift;
    my $p = $self->dir;
    $p =~ s!/[^/]+$!! or return $self;
    return new Dir(dir => $p);
}
```

Classes, objects (cont'd)

```
sub exists {
    my $self = shift;
    return -e $self->dir . '/.';
}
sub mkdir {
    my $self = shift;
    return 1 if $self->exists();
    $self->parent->mkdir();
    mkdir $self->dir;
}
1;
```

We can then create directory in an object way:

```
$ ls /tmp/pokus/hokus
ls: cannot access /tmp/pokus/hokus: No such file or directory
$ perl -MDir -e 'my $d = new Dir(dir => "/tmp/pokus/hokus"); $d->mkdir();'
$ ls /tmp/pokus/hokus
$
```

Subclasses

Some modules are designed to be used via subclassing.

In `UsersDBI.pm`, we have

```
package UsersDBI;
use strict;
use warnings FATAL => 'all';
use base 'Class::DBI';
UsersDBI->connection('dbi:Pg:dbname=perl301', 'perl301user', 'teslo',
    { RaiseError => 1, AutoCommit => 0 });
```

The file `Users.pm` contains

```
package Users;
use base 'UsersDBI';
Users->table('users');
Users->columns(All => ('login', 'password', 'uid'));
```

So far, everything is very much a declaration.

```
use Users ();
for my $row (Users->retrieve_all) {
    if (not exists $users{$row->login}) {
```

Subclasses (cont'd)

```
        print "Removing [@{[$row->login]}]\n";
        $row->delete();
        next;
    }
    {
    no warnings 'uninitialized';
    my $changed = 0;
    if ($row->uid ne $users{$row->login}[0]) {
        $row->uid($users{$row->login}[0]);
        $changed = 1;
    }
    if ($row->password ne $users{$row->login}[1]) {
        $row->password($users{$row->login}[1]);
        $changed = 1;
    }
```

Subclasses (cont'd)

```
    if ($changed) {
        print "Fixing [@{[$row->login]}]\n";
        $row->update();
    }
    }
    delete $users{$row->login};
}
for my $login (keys %users) {
    Users->insert({
        login => $login,
        password => $users{$login}[1],
        uid => $users{$login}[0]
    });
    print "Adding [$login]\n";
}

Users->dbi_commit();
```

Plain CGI

In `/var/www/cgi-bin/script.cgi`:

```
#!/usr/bin/perl -T
use strict;
use warnings FATAL => 'all';
print "Content-Type: text/plain\nPragma: no-cache\n\n";
print "This is the output of the CGI script.\n";
```

```
$ GET 'http://localhost/cgi-bin/script.cgi'
This is the output of the CGI script.
```

CGI::Simple

```
#!/usr/bin/perl -T
use strict;
use warnings FATAL => 'all';
use CGI::Simple ();
my $q = new CGI::Simple();
$q->no_cache(1);
print $q->header(-status => 200, -type => 'text/plain',
    'X-Header' => 'scripting');
print "This is the output of the CGI script.\n";
for ($q->param('id')) {
    print "You passed in id [$_]\n";
}
1;
```

CGI::Simple (cont'd)

```
$ GET -Se 'http://localhost/cgi-bin/script1.cgi?id=123;id=9'
Connection: close
Date: Thu, 15 Dec 2011 13:53:25 GMT
Pragma: no-cache
Server: Apache/2.2.17 (Fedora)
Content-Type: text/plain; charset=ISO-8859-1
Expires: Thu, 15 Dec 2011 13:53:25 GMT
Client-Date: Thu, 15 Dec 2011 13:53:25 GMT
Client-Peer: 127.0.0.1:80
Client-Response-Num: 1
Client-Transfer-Encoding: chunked
X-Header: scripting

This is the output of the CGI script.
You passed in id [123]
You passed in id [9]
```

```
$ echo -n id=13 | POST 'http://localhost/cgi-bin/script1.cgi'
This is the output of the CGI script.
You passed in id [13]
```

mod_perl

```
LoadModule perl_module modules/mod_perl.so
```

For quick move from CGI to mod_perl, you can use `ModPerl::Registry` to run unchanged script (that operate on STDIN and STDOUT)::

```
Alias /perl /var/www/perl
<Directory /var/www/perl>
    SetHandler perl-script
    PerlResponseHandler ModPerl::Registry
    PerlOptions +ParseHeaders
    Options +ExecCGI
</Directory>
```

```
$ GET 'http://localhost/perl/script1.cgi?id=100&px=1'
This is the output of the CGI script.
You passed in id [100]
```

Having `/perl-status` enabled helps when debugging.

mod_perl (cont'd)

For more serious work, write handlers.

```
package handler;
use Apache2::RequestIO ();
use Apache2::RequestRec ();
use Apache2::Request ();
sub handler {
    my $r = shift;
    $r->content_type('text/plain');
    $r->print("Testing handler\n");
    my $req = new Apache2::Request($r);
    for ($req->param('id')) {
        print "You passed in id [$_]\n";
    }
    return Apache2::OK;
}
1;
```

mod_perl (cont'd)

```
<Location /testh>
 SetHandler perl-script
 PerlResponseHandler handler
</Location>
```

When the modules is stored in say `/var/www/perllib/handler.pm`, configure with

```
PerlSwitches -I/var/www/perllib
```

```
$ echo -n id=99 | POST 'http://localhost/testh?id=4'
Testing handler
You passed in id [4]
You passed in id [99]
```

Self-assessment

- Go and create masterpieces.

Final questions?

No?

OKay. Thank you!