Intermediate Perl Scripting for Network Operators

John Kristoff jtk@cymru.com

overview

- We'll assume basic Perl competency
- We'll introduce intermediate Perl concepts and usage
- We'll learn by building a set of BGP-related tools
- We'll try to write portable scripts
- We'll use real world examples you can build upon
- We'll assume you'll be hacking while I talk
- We'll not be exhaustive since our time is limited
- We'll encourage you to write and share more tools
- Get this: http://www.cymru.com/jtk/code/nanog54.tar.gz

what we'll be covering

- Writing safe and maintainable code
- Modules and subroutines
- Common Perl idioms (I use)
- References
- I/O operations
- Database integration
- CGI

how I start most Perl scripts

```
#!/usr/bin/perl -T
use strict;
use warnings;
$| = 1;

# $Id: $
```

taint mode

- `-T` to enforce, `-t` for taint warnings
- A contrived, but illustrative example:

```
open my $fh, $ARGV[0];
```

- Then this would do what you think it does:

```
./unsafe.pl 'rm -rf / |'
```

- Untaint

```
my ($file)
  = $ARGV[0]
  =~ m{ \A ( [\w.-]+ ) \Z }xms;
```

routelogger.pl

- Receive and log BGP events from a peer
  i. Can send event messages to screen
  ii. Can send event messages to syslog
  iii. Can send UPDATE events to a database

example log messages

```
2012-01-31 17:00:00 UTC [6:341] BGP ↩
  session open from 192.0.2.1 AS64900

2012-01-31 17:00:00 UTC [6:380] ↩
  Announcement from 192.0.2.1 AS64900 ↩
  for 192.0.2.0/25 192.0.2.128/25 with ↩
  AS path 64900 64901, next hop ↩
  192.0.2.1 and origin 0
```

getting started with Net::BGP

- Pull in modules, prepare objects and peer

```
use Net::BGP::Peer;
use Net::BGP::Process;
my $bgp  = Net::BGP::Process->new;
my $peer = Net::BGP::Peer->new(
    %SESSION_CONFIG
);
$bgp->add_peer($peer);
$bgp->event_loop();
```

%SESSION_CONFIG hash

- Remember, a hash is just a key/value list
- Net::BGP::Peer Object requires peer config info

```
my %SESSION_CONFIG = (
     Start  => 1, # idle state if false
     ThisID => $local_addr,
     ThisAS => $local_asn,
     PeerID => $peer_addr,
     PeerAS => $peer_asn,
     Listen => 0, # no passive listen
                  # note last comma
);
```

get BGP peering config from cli

- A lot of people use Getopt::Long
- I tend to just use getopts like this:

```
use Getopt::Std;
use constant USAGE => “$0 [ options]
  Options:
  -l local_addr      required param
  -L local_asn       required param
  -r remote_addr     required param
  -R remote_asn      required param
  -h                 show this message
“;
getopts( 'l:L:r:R:h', \my %opts );
```

working with the %opts hash

- If the switch is set, it'll evaluate to true
- When set the `%opts` key value is 1 or arg value
- Examples of how we use switches and values

```
$opts{h} && die USAGE;
my $local_asn
  = untaint_addr( $opts{L} )
  || die USAGE;
```

sub untaint_asn;

```
sub untaint_asn {
  my $asn = shift || return;

  return
    if $asn !~ m{ \A \d{1,5} \Z }xms;

  # Net::BGP only supports 16-bit ASNs
  return
    if $asn < 0 || $asn > 65535;

  ($asn) = $asn =~ m{ \A (\d+) \Z }xms;

  return $asn;
}
```

acting on UPDATE messages

- Setup a callback handler

```
$peer->set_update_callback(
  \&callback_update
);

# ...

sub callback_update {
  my ( $peer, $update ) = @_;
  my $nlri_ref = $update->nlri;
  my $withdrawn_ref
    = $update->withdrawn;
```

references

- Like C pointers w/o the memory management

```
$s_ref = \$foo;  $a_ref = \@bar;
$h_ref = \%baz;  $c_ref = \&qux
```

- Dereferencing can look a bit ugly

```
$$s_ref = 'foo';  $$a_ref[0] = 'foo';
%$h_ref = ( foo => 'bar' );
```

- The arrow notation for arrays and hashes

```
$a_ref->[0] = 'foo';
$h_ref->{foo} = 'bar';
```

dereferencing the updates

- Prepend a `@` to the reference and use like an array

```
if ( scalar @$withdrawn_ref > 0 ) {

  my $prefix_list = "@$withdrawn_ref";

  my $message
    = sprintf
    "Withdrawal from %s AS%s for %s",
    $peer->peer_id,
    $peer->peer_as,
    $prefix_list;
```

generic stdout/syslog routine

- Handles messages to syslog and stdout
- This routine should “do the right thing”
- We use the standard syslog level macro names
- __LINE__ is the source code line called from

```
logit( LOG_INFO, __LINE__, $message );
```

logit initialization

```
sub logit {
  my ( $level, $line_num, $message )
    = @_;
  my $stamp = current_timestamp;

  $level    ||= LOG_WARNING;
  $line_num ||= 0;
  $message  ||= 'unspecified event';
```

passing subroutine arguments

- What is wrong with this?

```
foo( %bar, @baz, $qux );
sub foo { my %quux = shift; # . . . }
```

- Subroutine arguments are passed as a flat list
- Thus we often pass by reference instead of value
- Passing by reference can also be more economical

connecting to the database

```
my $dbh_ref;
if ( $opts{d} ) {
     $dbh_ref = db_connect(
          {
               dbhost => $opts{b},
               dbtype => $opts{i},
               dbport => $opts{p},
               Dbname => $opts{n},
               Dbuesr => $opts{u},
               Dbpass => $opts{a},
          }
     );
}
```

using named arguments

- Implemented as an anonymous hash reference
- Not always the prettiest, but Perl BP suggests it

```
sub db_connect {
  my ($arg_ref) = @_;
  my $db_type
    = $arg_ref->{dbtype} || 'Pg';
```

without DBI placeholders

- Variables inline to SQL statements are dangerous
- What if:

```
# $baz = '1; DELETE FROM foo'
my $sql =
 “SELECT * FROM foo WHERE bar = $baz”;
$dbh->prepare($sql);
$dbh->execute;
```

- Ouch

with DBI placeholders

- Placeholder values separated from the SQL
- Practically eliminates SQL injection attacks
- Fixed code:

```
# $baz = '1; DELETE FROM foo'
my $sql =
 'SELECT * FROM foo WHERE bar = ?';
$dbh->prepare($sql);
$dbh->execute($baz);
```

- This SQL will now likely just error out, phew

routerlogger perlcritic report

- `perlcritic -5` (OK)
- `perlcritic -4` (+6 violations)
  - Declare some vars local, don't use constant
- `perlcritic -3` (+23 violations)
  - Formatting, croak, reused vars, regex /x
- `perlcritic -2` (+38 violations)
  - POD, Readonly, regex, layout
- `perlcritic -1` (+19 violations)
  - Layout, regex, useless interpolation

routelogger-report.pl

- Summarize routelogger syslog events
  i. Total announcements and withdrawals
  ii. Most active prefixes
  iii. Alert on “golden networks”
  iv. Send output to STDOUT or email

example output

```
Total announcements: 15
Total withdrawals: 3
Most active prefixes (count, prefix)
  12    192.0.2.0/24
   7    192.0.2.0/25
   4    192.0.2.128/25
IPv4 golden net exact match update ↵
  Found: 192.0.2.0/128
```

getting started with Net::Patricia

- Pull in modules, prepare objects and gather data

```
use Net::Patricia;
my $pt_golden4;
my $pt_golden6;
if ( $opts{g} ) {
   $pt_golden4 = new Net::Patricia;
   $pt_golden6
     = new Net::Patricia(AF_INET6);
   parse_golden_nets( $opts{g} );
}
```

opening, reading and closing files

```
open( my $GOLDEN_FILE, '<', $filename )
  or die
  “Unable to open $filename: $!\n”;

while (
  defined( my $line = <$GOLDEN_FILE> )){

# . . .

}
close $GOLDEN_FILE
  or die
  “Unable to close $filename: $!\n”;
```

Readonly and compiled regex

- Recall log message format from routelogger.pl

```
Readonly my $UPDATE => qr{
   ( Announcement | Withdrawal )
  \s from \s
  (\S+) \s
  AS(\S+) \s for \s
  (.*?) (?: with | \Z )
}xms;
```

gather statistics

```
next if $line !~ /$UPDATE/;
my ( $type, $peer_id, $peer_as,
   $prefix_list ) = ( $1, $2, $3, $4 );

$type eq 'Announcement'
   ? $announce++
   : $withdrawal++;

my @prefixes = split /\s/, $prefix_list;
for my $prefix (@prefixes) {
   ${update}{$prefix}++;
}
```

compute top 10 hash key values

- Some use the <=> comparison operator

```
my $loop_counter = 1;
for my $prefix (
 sort { $update{$b} - $update{$a} }
 keys %update
) {
  push @results,
    “$update{$prefix}\t$prefix\n”;
  last if $loop_counter++ == 10;
}
```

sending email

- Build up output and stick into array `@results`

```
my $email = Email::Simple->create(
  header => [
    From    => $from,
    To      => $to,
    Subject => '### routelogger ...',
  ],
  body => join '', @results,
);
sendmail($email)
```

routeinjector.pl

- Update a peer with configured announcements
  i. Monitor a database table for updates
  ii. Send route changes as UPDATE to peer
  iii. Like routelogger.pl, but with route injection

similar to routelogger except for...

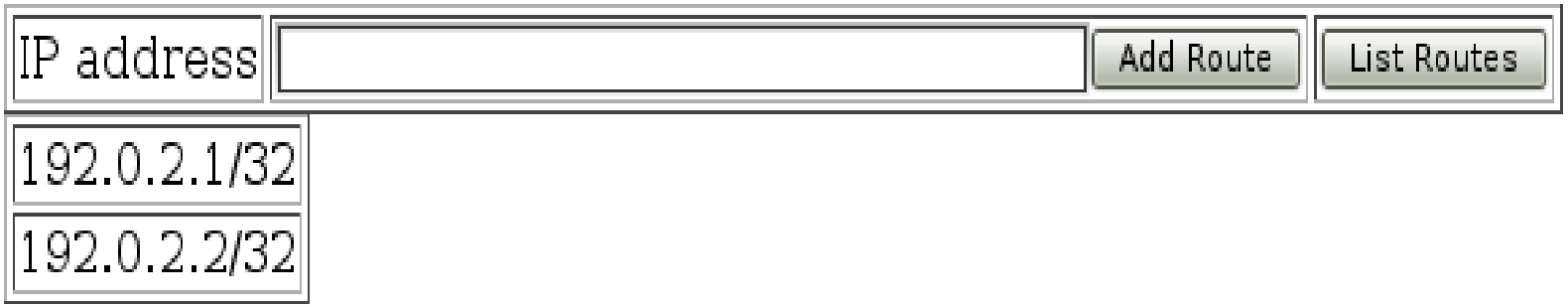
- Periodically see if an UPDATE is necessary

```
my $candidate_ref = get_routes();
if ($candidate_ref) {
  my $diff = Array::Diff->diff(
    $announce_ref, $candidate_ref );
  send_update( $diff->added,
    $diff->deleted );
}
else {
  send_update( undef, $announce_ref );
}
$announce_ref = $candidate_ref;
```

bhrs-admin.cgi

- Maintain a database of announcements via CGI
  i. Display configured announcements
  ii. Add new announcements
  iii. Limited CGI interface to route-injector.pl data

example CGI, pretty lame eh?



hardening tasks

```
delete @ENV{
  qw( IFS CDPATH ENV BASH_ENV )
};

$ENV{PATH}
  = '/bin:/usr/bin:/usr/local/bin';

$CGI::POST_MAX = 1024 * 16; # 16 KB

$CGI::DISABLE_UPLOADS = 1;
```

figuring out what to do next

```
my $page_action
   = param('.action') || 'default';

my %page_choice = (
   'add'     => \&add,
   'list'    => \&list,
   'default' => \&default,
);

put_page_top();
$page_choice{ $page_action }->();
put_page_bottom();
```

in English

- A HTTP POST carries a field named “.action”
- This field takes one of 3 values: add, list or default
- Page is built in part based on that field value, e.g.

```
print submit( -name => 'Add Route' );
print hidden(
     -name      => '.action',
     -value     => 'add',
     -override => 1,
);
```

building a page is a bit of an art

- CGI syntax is straightforward if not cumbersome
- Large CGI apps will take great care
- Consider more modern frameworks
  - e.g. Mason, Catalyst, CGI::Application
- Debugging is a little trickier
  - During dev, send errors to browser
  - tail -f the dev web server log

in closing

- “I don't know Perl, I know combat Perl”
- “Don't run this as root”
- Perl Best Practices, Damien Conway
- Please send questions, suggestions or scripts to:

jtk@cymru.com

PGP key 0xFFE85F5D

http://www.cymru.com/jtk/