Perl 101

Jan Pazdziora
Principal Software Engineer
Satellite Engineering, Red Hat

21th October 2011

Hello world

```
$ perl -e 'print "Hello world\n"'
Hello world
```

```
$ cat hello.pl
#!/usr/bin/perl
print "Hello world\n";
```

```
$ perl hello.pl
Hello world
```

```
$ chmod a+x hello.pl
$ ./hello.pl
Hello world
```

Variables (scalar)

```
$apples = 3; $oranges = '4 oranges';
print $apples + $oranges + plums + $pears, "\n";
```

```
7
```

Conversion of types and usage of uninitialized variables and functions works but you should *not* take advantage of them except for one-liners.

When we add `use strict;` to our code, we will get:

```
Global symbol "$apples" requires explicit package name at - line 2.
Global symbol "$oranges" requires explicit package name at - line 2.
Global symbol "$apples" requires explicit package name at - line 3.
Global symbol "$oranges" requires explicit package name at - line 3.
Global symbol "$pears" requires explicit package name at - line 3.
Bareword "plums" not allowed while "strict subs" in use at - line 3.
Execution of - aborted due to compilation errors.
```

Local variables declared

```
use strict;
my ($apples, $oranges) = (3, '4 oranges');
my $pears;
my $plums = 0;
my $total = $apples + $oranges + $plums + $pears;
print "Total: $total\n";
```

```
7
```

Even if all variables are declared as scoped local, we will get warnings in runtime when we `use warnings;` which we probably should for anything but the smallest programs.

```
Argument "4 oranges" isn't numeric in addition (+) at - line 6.
Use of uninitialized value $pears in addition (+) at - line 6.
Total: 7
```

Strict hello world

The first minimal Hello world program should really read:

```
#!/usr/bin/perl
use strict;
use warnings FATAL => 'all';
print "Hello world\n";
1;

=head1 NAME

Hello - greetings to the world.

=head1 SYNOPSIS

    hello

=head1 DESCRIPTION
```

Numeric vs. string operations

```
my $total = $apples + $oranges + $plums + $pears;
print "Total: $total\n";
```

```
7
```

```
my $total = $apples . $oranges . $plums . $pears;
print "Total: $total\n";
```

```
34 oranges
```

Reading input

```
my $i = 1;
while (defined(my $line = <>)) {
    chomp $line;
    print "$i: [", $line, "]\n";
    $i++;
}
```

```
$ ./numit.pl /etc/passwd /etc/group
1: [root:x:0:0:root:/root:/bin/bash]
2: [bin:x:1:1:bin:/bin:/sbin/nologin]
3: [daemon:x:2:2:daemon:/sbin:/sbin/nologin]
...
113: [kvm:x:36:qemu]
114: [qemu:x:107:]
115: [ldap:x:55:]
```

$_

```
my $i = 1;
while (<>) {
    chomp;
    print "$i: [$_]\n";
    $i++;
}
```

Other special global variables:

```
while (<>) {
    chomp $_;
    print "$.: [$_] in [$ARGV]\n";
}
```

```
1: [root:x:0:0:root:/root:/bin/bash] in [/etc/passwd]
2: [bin:x:1:1:bin:/bin:/sbin/nologin] in [/etc/passwd]
...
114: [qemu:x:107:] in [/etc/group]
115: [ldap:x:55:] in [/etc/group]
```

Looping from command line

```
while (<>) {
    print uc $_;
}
```

```
$ perl -lne 'print uc' /etc/passwd
ROOT:X:0:0:ROOT:/ROOT:/BIN/BASH
...
```

```
$ perl -MO=Deparse -Wlne 'print uc'
BEGIN { $^W = 1; }
BEGIN { $/ = "\n"; $\ = "\n"; }
use warnings;
LINE: while (defined($_ = <ARGV>)) {
    chomp $_;
    print uc $_;
}
-e syntax OK
```

Regular expressions

```
while (<>) {
    print if /^root:/;
}
```

```
while (defined(my $line = <>)) {
    print $line if $line =~ /^root:/;
}
```

```
10.23.89.12 - - [18/Oct/2011:16:53:53 +0200] "GET / HTTP/1.1" 302 274 ⏎
            "-" "Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.23)"
```

```
while (<>) {
    /^(\S+).*?"GET (\/\S*)/ and print "$1 $2\n";
}
```

Regular expressions (cont'd)

```
# Database name, or more precisely database service name.
db_name = prod

# Database user.
db_user = projectx

# Password for the database user.
db_password =
```

```
my $desc = '';
while (<>) {
    if (/^$/) { $desc = ""; next; }
    if (s/^#\s*//) { $desc .= $_; next }
    if (/^(\S+?)\s*=\s*(.*)/) {
        print "<dt>$1</dt>\n<dd>$desc</dd>\n";
        print qq!<dd>The default value is "$2"</dd>\n!;
        $desc = '';
    } else {
        warn "Unsupported config line $_"
    }
}
```

One-liners

```
$ perl -ne 'if (/^$/) { $desc = "" } elsif (s/^#\s*//) { $desc .= $_ }⏎
       elsif (/^(\S+?)\s*=/) { print "<dt>$1</dt>\n<dd>$desc</dd>\n" }'
```

```
# perl -i -pe 's!:/bin/false$!:/sbin/nologin!' /etc/passwd
```

```
$ perl -le 'print for 1001..1010'
```

Explicit open of files

```
local *FILE;
open FILE, '<', $name or die "Error reading $name: $!\n";
while (<FILE>) {
...
}
```

```
use IO::File ();
my $fh = new IO::File $name, 'r'
                      or die "Error reading $name: $!\n";
local *OUT;
open OUT, '>', "$name.out" or die;
while (defined(my $line = <$fh>)) {
    # process $line
    print OUT $line;
}
```

Explicit open of files (cont'd)

```
while (defined(my $line = $fh->getline)) {
...
}
```

Arrays

```
my $total = 0;
for (@ARGV) {
    $total += $_;
}
print $total, "\n";
```

```
$ ./sum 79234 9923 0 12 -234
88935
```

```
my $total = 0;
$total += $_ for @ARGV;
```

```
my $silent = 0;
if (@ARGV and $ARGV[0] eq '-s') {
    $silent = 1;
    shift @ARGV;
}
```

List context

```
print <$fh>;
```

```
my $i = 0;
print map { ++$i . ': ' . $_ } <$fh>;
```

```
$ perl -le 'print localtime'
1153151891112290 1
$ perl -le 'print join " ", localtime'
11 53 15 18 9 111 2 290 1
$ perl -le 'print scalar localtime'
Tue Oct 18 15:53:11 2011
```

```
my @time = localtime;
printf "%02d:%02d:%02d\n", $time[2], $time[1], $time[0];
```

```
16:00:45
```

List context (cont'd)

```
printf "%02d:%02d:%02d\n", (localtime)[2, 1, 0];
```

```
my ($name, $pass, $uid) = split /:/, $_;
```

```
my ($name, $uid) = (split /:/)[0, 2];
```

```
my ($name, $uid) = /^(.+?):.*?:(.+?):/;
```

Hashes (associative arrays)

```
$ /usr/bin/which ls httpd
/bin/ls
/usr/sbin/httpd
```

```
my @path = split /:/, $ENV{PATH};
for my $a (@ARGV) {
    for my $p (@path) {
        if (-f "$p/$a") {
            print "$p/$a\n"; last;
        }
    }
}
```

```
$ ./which.pl ls httpd
/bin/ls
/usr/sbin/httpd
```

Hashes (associative arrays) (cont'd)

```
local *CONF;
open CONF, '<', '/etc/projectx.conf' or die;
my %config;
while (<CONF>) {
    next if /^#/;
    if (/^\s*(\S+?)\s*=\s*(.+)/) {
        $config{$1} = $2;
    }
}
close CONF;

use Data::Dumper;
print Dumper \%config;
```

Functions

```
sub add {
    my $total = 0;
    for (@_) {
        $total += $_;
    }
    return $total;
}
my $soucet = add(3, 5, 12);
my $soucet1 = add 3, 5, 12;
```

```
sub process {
    my ($dbh, $message) = @_;
    # ...
    $message =~ s/line/Line/;
    # ...
}
```

Functions (cont'd)

```
my %config;
sub parse_config_file {
    my $filename = shift;
    local *CONF;
    open CONF, '<', $filename or die "Error reading $filename: $!\n";
    while (<CONF>) {
        next if /^#|^\s*$/;
        if (/^\s*(\S+?)\s*=\s*(.*)/) {
            $config{$1} = $2;
        } elsif (/^\s*include\s+(.+)/) {
            parse_config_file($1);
        } else {
            chomp;
            warn "Wrong syntax [$_] at $filename:$.\n";
        }
    }
    close CONF;
}
parse_config_file($ENV{'PROJECTX_CONF'} // '/etc/projectx.conf');
```

References

```
sub parse_config_file {
    my $config = ( ref $_[0] ? shift : {} );
    my $filename = shift;
    local *CONF;
    open CONF, '<', $filename or die "Error reading $filename: $!\n";
    while (<CONF>) {
        next if /^#|^\s*$/;
        if (/^\s*(\S+?)\s*=\s*(.*)/) {
            $config->{$1} = $2;
        } elsif (/^\s*include\s+(.+)/) {
            parse_config_file($config, $1);
        } else {
            chomp;
            warn "Wrong syntax [$_] at $filename:$.\n";
        }
    }
    close CONF;
    return $config;
}
my $config = parse_config_file('/etc/projectx.conf');
```

Using objects

```
use File::Temp ();
my $tempfile = new File::Temp(
    'DIR' => '/tmp',
    'TEMPLATE' => 'projectxXXXX',
);
my $filename = $tempfile->filename;
print "Filename: ", $filename, "\n";
print $tempfile "Test\n";
$tempfile->flush();
print "Length: @{[ -s $filename ]}\n";
$tempfile->close;
system "cat $filename";
$tempfile = undef;
if (not open FILE, $filename) {
    print "File was removed\n";
}
```

Input data

```
$ cat input8
maličký ježek
$ od -tx1 input8
0000000 6d 61 6c 69 c4 8d 6b c3 bd 20 6a 65 c5 be 65 6b
0000020 0a
0000021
$ perl -ne 'print length, "\n"' input8
17
$ perl -lpe 's/\W/./g' input8
mali..k...je..ek
$ perl -ne 'print uc' input8
MALIčKý JEžEK
```

```
$ iconv --from-code UTF-8 --to-code ISO-8859-2 input8 > input2
$ od -tx1 input2
0000000 6d 61 6c 69 e8 6b fd 20 6a 65 be 65 6b 0a
0000016
$ perl -lpe 's/\W/./g' input2
mali.k..je.ek
$ perl -ne 'print uc' input2
MALI?K? JE?EK
```

Use characters internally

```
$ perl -ne 'print uc' input8
MALIčKý JEžEK
$ perl -CSD -ne 'print uc' input8
MALIČKÝ JEŽEK
```

```
$ perl -ne 'print uc' input2
MALI?K? JE?EK
$ perl -ne 'print uc' input2 | iconv -f ISO-8859-2 -t UTF-8
MALIčKý JEžEK
$ perl -e 'print uc <>' input2 | iconv -f ISO-8859-2 -t UTF-8
MALIčKý JEžEK
$ LC_CTYPE=cs_CZ perl -e 'use open ":locale"; print uc <>' input2 \
    | iconv -f ISO-8859-2 -t UTF-8
MALIČKÝ JEŽEK
$ LC_CTYPE=cs_CZ perl -Mopen=:locale -e 'print uc <>' input2 \
    | iconv -f ISO-8859-2 -t UTF-8
MALIČKÝ JEŽEK
$ LC_CTYPE=cs_CZ perl -CSD -Mopen=IN,:locale -e 'print uc <>' input2
MALIČKÝ JEŽEK
```

Specify encoding per filehandle

```
use open ':std', ':utf8';
local *FILE2;
open FILE2, '<:encoding(iso-8859-2)', 'input2' or die;
while (<FILE2>) {
    print uc;
}
close FILE2;
```

```
local *FILE2;
open FILE2, '<', 'input2' or die;
binmode FILE2, ':encoding(iso-8859-2)';
binmode STDOUT, ':utf8';
while (<FILE2>) {
    print uc;
}
close FILE2;
```

Change encoding mid-write

```
use warnings;
use open ':std', ':utf8';
while (<>) {
    chomp;
    binmode STDOUT, ':utf8';
    print "UTF-8: $_; ";
    binmode STDOUT, ':encoding(utf7)';
    print "UTF-7: $_\n";
}
```

```
$ ./utf8utf7 input8
UTF-8: maličký ježek; UTF-7: mali+AQ0-k+AP0- je+AX4-ek
```

UTF-8 in source

```
use utf8;
my $var = "maličký ježek";
print length($var), ", ", $var, "\n";
```

```
$ ./printfutf8
Wide character in print at - line 3.
13, maličký ježek
```

```
$ perl -CS ./printfutf8
13, maličký ježek
```

Or `use open ":utf8", ":std"`.

Conversions

```
use Encode;
use utf8;
open FILE, '<:bytes', 'input2';
binmode STDOUT;
while (<FILE>) {
    my $u = "Výstup: " . Encode::decode('iso-8859-2', $_);
    print Encode::encode('windows-1250', $u);
}
```

```
use Encode;
use utf8;
open FILE, '<:bytes', 'input2';
binmode STDOUT;
while (<FILE>) {
    my $u = "Výstup: " . Encode::decode('iso-8859-2', $_);
    while ($u ne '') {
        print Encode::encode('iso-8859-1', $u, Encode::FB_QUIET);
        $u =~ s/^./-/;
    }
}
```

Self-assessment

```
Return-path: <bounce-user-23492fw90wweuw49@gmail.com>
Envelope-to: peter@example.com
Delivery-date: Thu, 20 Oct 2011 08:53:51 +0000
Date: Thu, 20 Oct 2011 08:53:51 +0000
From: John User <user@gmail.com>
To: peter@example.com
Subject: =?utf-8?B?UMWZw6Fuw60=?=
Message-ID: <20111020085350.GB7116@relay4.gmail.com>
MIME-Version: 1.0
Content-Type: text/plain; charset=utf-8
Content-Disposition: inline
Content-Transfer-Encoding: 8bit
User-Agent: Web-based-super-mail/1.5.20 (2009-12-10)

Všechno nejlepší k narozeninám.

Honza
```

Self-assessment (cont'd)

Write a Perl script which will read email message (see previous example) from standard input and print text representation of the email to standard output, suitable for example for SMS notifications. Use UTF-8 as the output character set.

Example output:

```
F: user@gmail.com S: Přání
Všechno nejlepší k narozeninám.
Honza
```

Also support different input charsets. Also support multipart emails (show content of text/plain part). Also support HTML-only emails (show text representation of those).

Make it perfect.

Final questions?

No?

OKay. Thank you!