Perl Programming

A Crash Course

Dr Russ Ross

Dixie State College—Computer and Information Technologies

September 16, 2010

UID to username

This script finds the username for a given UID:

```
#!/usr/bin/perl -w

use strict;

sub getuname {
    @_ != 2 and die "Wrong number of arguments";

    my $uid = shift;
    my $fname = shift;

    open FP, "<$fname" or die "Unable to open $fname: $!\n";
    while (<FP>) {
        my @items = split ':', $_;
        return $items[0] if $items[2] == $uid;
    }
    die "Not found";
}

print getuname(1001, "/etc/passwd") . "\n";
```

Perl data types

Perl types are distinguished by their shape. We will consider the following principal data types:

- `$`Scalars (numbers, strings, references)
- `@`Arrays (lists of scalars)
- `%`Hashes (maps from strings to scalars)
- `$`, `@`, `%`, get it?
- File handles

There are others, but these are the main types we will focus on.

Scalars

A scalar is a single value. Scalars are named with a dollar sign (`$`). Note that the dollar sign names a value, not a variable. Some examples:

- `0`, `1.5`, `6.04e23`, `1_000_000`
- `$x`, `$myarray[5]`
- 'Quoted string', "Double quoted string"

Basic operators are the same as C/C++/Java:

- `$a = 1 + 2`
- `$a = 5 % 2`
- `++$a`, `$a++`
- `$a = 5 + 3 * 8`

Strings

- Double-quoted strings are interpolated:

```perl
my $name = 'Russ';
my $shoesize = 12;
print "$name has an IQ of $shoesize\n";
```

- Single-quoted strings are not interpolated.
- Strings are automatically converted to numbers and vice versa (Perl considers them to be the same thing).
- + adds two numbers, . concatenates two strings:

```perl
print '7' + '4';
print 'Hello' + ' world';
print '7' . '4';
print 'Hello' . ' world';
```

Arrays

Arrays are named with an `@` sign:

```
my @rodents;
```

Array literals:

```
@rodents = ('mouse', 'rat', 'squirrel', 'gerbil');
```

Arrays are always flat:

```
@rodents = (('mouse', 'rat'), (('squirrel'), 'gerbil'));
```

Arrays are copied by default:

```
@a = @b; pop @a; #@b is unaffected
```

Array operations

- `push @rodents, 'mole';`
- `print "I know about @rodents\n";`
- `my $first = shift @rodents;`
- `unshift @rodents, 'chipmunk';`
- `pop @rodents;`
- `print "I like $rodents[1]s best\n";`
- `print "The last kind is $rodents[-1]\n";`
- `print "@rodents";`
- `print "@rodents[0..2]";`
- `print "@rodents[0,2]";`

Context

Perl expressions are always evaluated in some **context.** If an array value is expected, an expression may give a different result than the same expression if a scalar is expected:

```
print @rodents;

print 0 + @rodents;
```

Greenfly

Greenfly can reproduce asexually. After one week of life a lone female can produce either offspring a day. Starting at the beginning of day 1 with a single mature female, how many greefly could there be by the end of day 28? It may be assumed that:

- There are no deaths
- All offspring are females

Note that at the end of:

- day 1 there will be 9 greenfly (original + 8 offspring)
- day 7 there will be 57 greenfly (original + 8$\times$7 offspring)
- day 8 there will be 129 greenfly (original + 8$\times$8 offspring + 64 offspring from the daughters produced on day 1).

Greenfly problem

This can be solved with:

```
#!/usr/bin/perl -w

use strict;

my @flies = (0, 0, 0, 0, 0, 0, 1);

foreach (1..28) {
    my $mature = pop @flies;
    unshift @flies, $mature * 8;
    $flies[-1] += $mature;
}

my $total = 0;
$total += $_ foreach @flies;
print "total after 28 days is $total\n"
```

Hashes

Associate arrays, or hashes, map keys to values, and are a basic datatype in Perl:

```perl
my %grades = ('Kimberly' => 'B', 'Kay' => 'C+', 'Jonathan' => 'D');
$grades{'Kimberly'} .= '-';
$grades{'Kay'} = substr($grades{'Kay'}, 0, 1);
```

Conditionals

If statements:

```
if ($x > 7) {
    print "$x\n";
} elsif ($x < 3) {
    print "Boo\n";
} else {
    print "Yeah";
}

unless ($x > 5) {
    # la la la
}
```

The braces are always required, except when using the postfix form:

```
$x++ if $s == 'inc';
print 'I eat steak' unless $type == 'vegan';
```

Loops

A variety of loop types:

```
for (my $i = 0; $i < 10; $i++) { print "$i\n"; }
foreach my $i (1, 2, 3, 4) { print "$i\n"; }
for (1..10) { print; }
while ($x < 7) { $x++; }
until ($x > 8) { $x++; }
print $i++ while $i < 10;

# loop over an array
foreach my $elt (@rodents) { print "$elt\n"; }

# loop over values in a hash
while (($k, $v) = each %myhash) {
    print "$k = $v\n";
}
```

Subroutines

Functions in Perl are called subroutines:

```
sub bigger {
    my ($a, $b) = @_;
    return $a > $b ? $a : $b;
    # or $_[0] > $_[1] ? $_[0] : $_[1];
}
```

Parameters are not declared. They all arrive in an array called `@_`.

The last value computed in a subroutine is automatically returned if `return` is omitted.

Parentheses are optional when calling a subroutine, unless it causes ambiguity:

```
($a, $b) = (5, 2);
my $x = bigger $a, $b;
```

Implicit variables

In many places, you can leave out a parameter and Perl will supply a default value for you. Usually this is `$_` for scalars and `@_` for arrays:

```
open FP, "<input.txt" or die;
while (<FP>) {
    print;
}

sub rev {
    my @out;
    unshift @out, $_ foreach @_;
    @out
}
```

Regular expressions

Regular expressions are integrated into Perl. They are a variation of POSIX regular expressions that are now used by most other languages as well:

```
while (<>) {
    if (m/^DTITLE= *(.*?) *\/ *(.*?) *$/) {
        ($artist, $album) = ($1, $2);
    } elsif (m/^DYEAR= *(.*?) *$/) {
        $year = $1;
    } elsif (m/^DGENRE= *(.*?) *$/) {
        $genre = $1;
    } elsif(m/^TTITLE(\d+)= *(.*?) *$/) {
        $tracks[$1 + 1] = $2;
    }
}
```

Books

- *Learning Perl* is a good way to learn Perl if you already know how to program and are comfortable in Linux.
- *Programming Perl* is the standard text on the language, and is quite readable as well (I learned it by reading this book).
- *Perl Cookbook* has snippets of code to do many of the most common tasks. Reading those examples is a good way to get a feel for how the language should actually be used.

Above average

```perl
sub mean {
    my $sum = 0;
    $sum += $_ foreach @_;
    return $sum / @_;
}

sub above_average {
    my $mean = mean @_;
    my @above = ();
    $_ > $mean and push @above, $_ foreach @_;
    @above;
}

my @fred = &above_average(1..10);
print "\@fred is @fred\n";
print "(should be 6 7 8 9 10)\n";
my @barney = &above_average(100, 1..10);
print "\@barney is @barney\n";
print "(should be just 100)\n";
```

Calculator

```
for (;;) {
    print "cmnd: ";
    chomp(my $op = <STDIN>);
    last if $op eq 'quit';

    if ($op ne '+' && $op ne '-' && $op ne '*' && $op ne '/') {
        print "Unknown command: $op\n";
        print "Valid commands: '+', '-', '*', '/', or 'quit'\n\n";
        next;
    }
    print "arg1: ";
    chomp(my $arg1 = <STDIN>);
    print "arg2: ";
    chomp(my $arg2 = <STDIN>);

    my $result;
    $op eq '+' and $result = $arg1 + $arg2;
    $op eq '-' and $result = $arg1 - $arg2;
    $op eq '*' and $result = $arg1 * $arg2;
    $op eq '/' and $result = $arg1 / $arg2;
    print "$arg1 $op $arg2 = $result\n\n";
}
```

Sort middle

This code sorts the middle letters in each word of input.

```
sub sort_letters {
    my @letters = split //, shift;
    join '', sort @letters;
}

while (<>) {
    s/([A-Za-z])([A-Za-z]+)([A-Za-z])/$1.sort_letters($2).$3/ge;
    print;
}
```

Find PID

Find the PID of a process by name:

```perl
die "Usage: $0 <name>\n" unless @ARGV == 1;
my $target = shift @ARGV;

my $list = `ps ax`;
my @lines = split /\n/, $list;

#print scalar(@lines);

foreach my $line (@lines) {
    $line =~ s/^\s+//;
    my @fields = split /\s+/, $line;
    print $fields[0] . "\n" if $fields[4] eq $target;
}
```

Matrix transposition

```
use Data::Dumper;
sub readMatrix {
    my $m = [];
    while (<STDIN>) {
        chomp;
        last if m/^\s*$/;
        m/^\[ *(.*?) *\]$/ or die;
        push @$m, [ split / +/, $1 ];
    }
    $m; }
sub printMatrix {
    my $m = shift;
    print "[ " . (join ' ', @$_) . " ]\n" foreach @$m; }
sub transposeMatrix {
    my $m = shift;
    my $result = [];
    for (my $y = 0; $y < @$m; $y++) {
        my $row = $m->[$y];
        for (my $x = 0; $x < @$row; $x++) {
            $result->[$x]->[$y] = $m->[$y]->[$x];
        }}
    $result; }
```

Word count

```perl
#!/usr/bin/perl -w

use strict;

my $count = 0;

while (<>) {
    my @words = split /\s+/, $_;
    $count += @words;
}

print "$count\n";
```

Prime number sieve

```perl
my $target = <STDIN>;

chomp $target;

my @sieve = ();

my $limit = int(sqrt($target));

for (my $i = 2; $i < $limit; $i++) {
    unless (defined($sieve[$i])) {
        for (my $j = $i * 2; $j <= $target; $j += $i) {
            $sieve[$j] = 1;
        }
    }
}

for (my $i = 2; $i <= $target; $i++) {
    print "$i is prime\n" unless defined $sieve[$i];
}
```