Perl Tutorial

Dr. Charles Cavanaugh

3-14-2012

https://tigerbytes2.lsu.edu/users/ccav/perltut/

Overview

- Basic syntax and semantics
- Searching and replacing text
- Working with CSV files
- Accessing databases

For More Information

- perldoc perlintro & perlstyle – introduction & style
- perldoc perlsyn – syntax
- perldoc perldata & perlvar – data types & variables
- perldoc perlop & perlfunc – operators & functions
- perldoc perlopentut – files & I/O
- perldoc perlrequick – regular expressions
- perldoc perlsub – subroutines
- perldoc perlmod – modules

Syntax & Semantics

- Perl program header:

  #!/usr/bin/perl

  use strict;

  use warnings;
- Statements end with a semicolon like C.
- Whitespace ignored except in quoted strings.
- Strings may be quoted with single or double quotes; single causes everything within to be printed as-is.

Syntax & Semantics

- Unlike C, parentheses are generally optional in functions: print(“Hello\n”); same as print “Hello\n”;
- “{” and “}” enclose blocks as in C.
- Arrays are zero-indexed as in C.
- Comments: begin with “#” as in Bash shell scripts.
- “\n” = newline, “\t” = tab
- =, +, -, *, /, ==, !=, <, >, <=, >= work as expected
- &&, ||, ! work as expected

String Comparisons

- Strings are compared differently:

  “eq” (equal)

  “ne” (not equal)

  “lt” (less than)

  “gt” (greater than)

  “le” (less than or equal)

  “ge” (greater than or equal)
- E.g.: if ($response eq “Y” || $response eq “y”) {...}

Data Types: Scalars

- Scalars = normal single-value variables OR strings
- Examples:

```
my $sum = 0;
my $a = 1;
$sum += $a;
my $prompt = “\$”;
print $prompt.“\n”;
```

Data Types: Arrays

- Arrays = lists of values, may be mixed
- my @animals = (“camel”, “llama”, “owl”);

  my @numbers = (23, 42, 69);

  my @mixed = (“camel”, 42, 1.23);

  print $animals[1];

  print $animals;

  print @animals[1..$#animals];
- my @sorted = sort @animals;

Data Types: Hashes

- Key/value pairs
- my %fruit_color = (

  apple => “red”,

  banana => “yellow”);

  print $fruit_color{“apple”};
- Array of just keys: keys %fruit_color

  Just values: values %fruit_color

Variable Scoping

```
my $var = “value”; #creates block-scoped variable
$v2 = “whatever”; #creates global variable
my $x = 2;
if ( true ) {
    my $x = 1;
}
print $x;
```

Variable Scoping

```
my $var = “value”; #creates block-scoped variable
$v2 = “whatever”; #creates global variable
my $x = 2;
if ( true ) {
    $x = 1;
}
print $x;
```

Conditionals

- if (condition) {

  ...

  } elsif (other condition) {

  ...

  } else {

  ...

  }
- unless (condition) { ... } # if (!condition) { ... }

Looping

- while (condition) {

  ...

  }
- until (condition) { ... } # while (!condition) { ... }
- for (initialize ; test ; increment) { ... }# similar to C

  for (my $i = 0; $i < 10; $i++) { print $i.“\n”; }
- foreach: see next slide

Looping: foreach

```
my @array = (“cherry”, “strawberry”, “pretzel”);
foreach (@array) {
    print “$_\n”;
}
```

- $_ is the current value or line
- my @array=(1,2,3);

  foreach my $n (@array) { print $n."\n" }

foreach with Hashes

```
my %hash=(a=>1,b=>2,c=>3);
foreach my $key (keys %hash) {
    print $hash{$key}."\n"
}
```

- Hashes are not sorted in any particular order!

Files & I/O

- my $filename = “input.txt”;

  open(my $filehandle, “<”, $filename) or die $!;
- my $filename = "iris.dat";

  open(my $filehandle, "<", $filename) or die "$!: $filename";

  No such file or directory: iris.dat at - line 2.
- my $line = <$filehandle>; #reads a line
- my @lines = <$filehandle>; #reads all lines into array

Typical Line-by-Line Reading

- my $lines = 0;

  while (<$filehandle>) {

  $lines++;

  print $lines. “:” . $_ . “\n”;

  }
- Close file when finished (best practice):

  close $filehandle;

Output or Append to File

- open(my $filehandle, “>”, “out.txt”);

  print $filehandle “This is a line.\n”;

  print $filehandle “This is another line.\n”;

  close $filehandle;
- open(my $filehandle, “>>”, “out.txt”); #append

  print $filehandle “line 3\n”;

  close $filehandle;

Regular Expressions

- Major part of learning Perl. See perlrequick for help.
- ...

```
while(<$in>) {
    if (/foo/) { ... } # true if $_ contains “foo”
    if ($_ =~ /foo/) { ... } # same (match operator)
    $new =~ s/foo/bar/; # replace 1st “foo” with “bar”
    $new =~ s/foo/bar/g; # replace all “foo” w/ “bar”
}
```

Special Characters in Regular Expressions

Character(s)	Meaning
.	any single character
\s	a whitespace character
\S	a non-whitespace character
\d or [0-9]	a digit
\w or [a-zA-Z0-9_]	a word character
\D or [^0-9]	a non-digit
[-\(\)0-9]	a hyphen, (,), or digit
... and many others	

Quantifiers in Regular Expressions

Quantifier	Meaning
*	zero or more of what's before *
+	at least one of what's before +
?	at most one of what's before ?
{3}	exactly three of what's before {
{3,6}	three to six of what's before {
{3,}	at least three of what's before {

Positional Specifier	Meaning
^	match at start of string
$	match at end of string

Example

- Print non-blank lines read from STDIN:

```
while (<>) {
    next if /^$/; # continue to next iteration if blank
    print; # prints $_ by default
}
```

Simple Parsing with Regular Expressions

- Parentheses capture matching parts of regexp
- Use what's captured with $1, $2, etc.
- if ($email =~ /([^@]+)@(.+)/) {

  print “username = $1\n”;

  print “hostname = $2\n”;}
- $time =~ /(\d\d):(\d\d):(\d\d)/; # match hh:mm:ss

  $hours = $1; $minutes = $2; $seconds = $3;
- ($hours, $minutes, $seconds) =

  ($time =~ /(\d\d):(\d\d):(\d\d)/);

Subroutines

- Definition:

```
sub square { # args are in @_
    my $num = shift;
    my $result = $num * $num;
    return $result;
}
```

- Usage:

```
$sq = square(8);
```

Subroutines with Multiple Arguments

- Definition:

```
sub printmulti { # args are in @_
    my ($string, $times) = @_;
    for (my $i=0; $i<$times; $i++) {
        print $string;
    }
}
```

- Usage: printmulti “*”, 8; #or printmulti (“*”, 8);

Modules

- Add functionality to Perl
- Help on a module: perldoc Module::Name

  Example: perldoc Text::CSV
- Installing a module:

  perl -MCPAN -e 'install Module::Name'
- Another method:

  First do this: cpan App::cpanminus

  Thereafter: cpanm Module::Name

Using Modules

- In Perl script:

  use Module::Name;
- Example:

  use Text::CSV;

Searching and Replacing in Files

- perl -p -i -e 's/original text/replacement text/g' file
- Warning: replaces in the original file! (i means in-place)
- perl -p -i.bak -e ... does same but saves backup as “file.bak”.
- perl -p -e ... > newfile.txt outputs to a new file without altering original.
- -p = while (<>) { ... # your script } continue { print or die "-p destination: $!\n";}
- Useful: www.softpanorama.org/Scripting/Perlorama/perl_in_command_line.shtml

Working with CSV Files

- CSV files = Comma-Separated Value text file
- Commas delimit the values, w/ or w/o headers:

  SKU, Description, Price

  123, “Folding Chair with Cup Holder”, 10.00
- May be tab-delimited instead
- Common extensions: .csv and .txt

  .csv generally for comma-separated files

  .txt generally for tab-delimited files

Text::CSV in Perl

```
my $file = 'prospects.csv';
my $csv = Text::CSV->new();
open (my $fh, “<”, $file) or die $!;
while (<$fh>) {
    if($csv->parse($_)) {
        my @columns = $csv->fields();
        print join(“|”,@columns) . “\n”; } }
close $fh;
```

- Useful: http://perlmeme.org/tutorials/parsing_csv.html

Accessing Databases in Perl

- Useful: http://perlmeme.org/tutorials/connect_to_db.html
- Also: perldoc DBI
- At beginning of script: use DBI;
- Connect (for example):

```
my $user = "";

my $password = "";

my $dbh = DBI->connect("DBI:$driver:$database", $user, $password, ) or die $DBI::errstr;
```

Simple SQL Statement Execution

- $dbh -> do(“

  INSERT INTO people_i_know(name, age, pet)

  VALUES ('Carolyn',25,null),

  ('Steve',23,'cat'),

  ('Melissa',24,'dog'),

  ('Ritchie',24,'rabbit');

  ”) or die $dbh->errstr;

Better SQL Statement Execution Using Prepared Statement

- $sth = $dbh -> prepare(“

  INSERT INTO people_i_know(name, age, pet)

  VALUES (?, ?, ?)

  ”) or die $dbh->errstr;

  $sth->execute('Carolyn',25,null) or die $dbh->errstr;

  $sth->execute('Steve',23,'cat') or die $dbh->errstr;

  $sth->execute('Melissa',24,'dog') or die $dbh->errstr;

  $sth->execute('Ritchie',24,'rabbit') or die $dbh->errstr;

SQL SELECT Statement Using Prepared Statement

- my $sth = $dbh->prepare("

```
SELECT name, age, pet
 FROM people_i_know
WHERE age > ?
```

  ") or die $dbh->errstr;
- How to fetch values? Read on...

Fetching Values (Preferred Method)

- $sth->execute(23) or die $dbh->errstr;

  while (my $hash_ref = $sth->fetchrow_hashref) {

  print $hash_ref->{name}, " is ", $hash_ref->{age},

  " years old, and has a " , $hash_ref->{pet}, "\n";

  }

Conclusion

- Perl is a “Swiss Army Knife” of programming languages.
- Perl is highly convenient for munging large files.
- Perl has many modules in CPAN (www.cpan.org).
- Perl maxims:
  - "There's more than one way to do it" (TMTOWTDI)
  - "Perl makes easy things easy and hard things possible."

Exercise

- Using skeleton file exercise.pl and text file phone.csv:
  - Count the number of phones manufactured by Apple
  - Change all instances of “iOS” to “iPhoneOS”
  - Sort list by manufacturer and print list of manufs.
  - Challenge: count number of different manufacturers