

Introduction to Perl

Deniz Yuret

*Based on the perlintro man page by Kirilly "Skua" Robert

Outline

- > Perl magic
- > Details
 - » Running perl
 - » File I/O
 - » Regular expressions
 - » Scalars, Arrays, Hashes
 - » Subroutines
- > More perl magic

Perl magic

- > Printing lines that match a pattern

```
perl -ne 'print if /cat/' foo.txt
```

Perl magic

- > Cleaning HTML tags

```
perl -pe 's/<. +?>//g;' foo.html
```

Perl magic

- > Substituting strings in many files

```
perl -pi -e 's/foo/bar/g' *.*
```

Perl magic

- > Printing only the first occurrence of each line

```
perl -ne  
'print unless $seen{$_}++' foo
```

Perl magic

> Finding prime numbers

```
perl -le '(1x$_)!~/^(11+)\1+$/ &&
print for 2..100'
```

Perl Design Principle

There is more than one way to do it

TIMTOWTDI

Running perl 1: Script file

```
> more script.pl
print "Hello world";
```

```
> perl script.pl
Hello world
```

Running perl 2: Executable script file

```
> more script.pl
#!/usr/bin/perl
print "Hello world";
```

```
> chmod +x script.pl
```

```
> ./script.pl
Hello world
```

Running perl 3: Command line

```
> perl -e 'print "Hello world"'
Hello world
```

(-e means execute the following code)

Feeding perl input 1: Standard input

```
> perl test.pl < foo.txt
```

('<' is the standard unix mechanism for sending a file to stdin i.e. as if someone typed it)

Feeding perl input 2:
Fake standard input

```
> perl test.pl foo.txt
```

Feeding perl input 3:
File open

```
open(FP, "foo.txt");  
while(<FP>) {  
    tr/A-Z/a-z/;  
    print;  
}  
close FP;
```

Feeding perl input 4:
Opening and closing

```
> open(INFILE, "input.txt");  
> open(OUTFILE, ">output.txt");  
> my $line = <INFILE>;  
> my @lines = <INFILE>;  
> print OUTFILE @lines;  
> close(INFILE);  
> close(OUTFILE);
```

Processing input lines 1:
Read them to a variable

```
> perl -e  
  
'while($a = <>) {  
    $a =~ tr/A-Z/a-z/;  
    print $a; }'  
  
foo.txt
```

Processing input lines 2:
Use the default variable

```
> perl -e  
  
'while(<>) {  
    tr/A-Z/a-z/;  
    print; }'  
  
foo.txt
```

Processing input lines 3:
Use the -n switch

```
> perl -ne  
  
'tr/A-Z/a-z/;  
print;'  
  
foo.txt
```

Processing input lines 4:
Use the `-p` switch

```
> perl -pe  
  
 'tr/A-Z/a-z/;'  
  
foo.txt
```

Processing input tokens 1:
Splitting to a variable

```
perl -ne  
  '@a = split(/\s+/, $_);  
  print "$a[1]\n";'  
foo.txt
```

Processing input tokens 2:
Using default args

```
perl -ne  
  '@a = split;  
  print "$a[1]\n";'  
foo.txt
```

Processing input tokens 3:
Using the default array

```
perl -ne  
  'split;  
  print "$_[1]\n";'  
foo.txt
```

Processing input tokens 4:
Using the `-a` switch

```
perl -ane  
  'print "$F[1]\n";'  
foo.txt
```

Processing input tokens 5:
Using the `-l` switch

```
perl -lane  
  'print $F[1];'  
foo.txt
```

Modifying files in place
Using the -i switch

```
> perl -pi -e 's/foo/bar/' baz.txt  
  
(Changes baz.txt)
```

Running perl:
Some command line options

```
> -e execute  
> -n execute for every input line  
> -p like -n with a final print  
> -a auto-split into @F array  
> -l auto add/remove newlines  
> -i modify file in place  
> -C handle utf8 properly
```

Perl syntax:
Statements end with ;

```
> print "Hello world";  
» Hello world
```

Perl syntax:
Comments start with #

```
> # This is a comment  
  
»
```

Perl syntax:
Whitespace is irrelevant

```
> print  
  "Hello world"  
  ;  
  
» Hello world
```

Perl syntax:
...except inside quotes

```
> print "Hello  
world";  
  
» Hello  
world
```

Perl syntax:
Double or single quotes

```
> print "Hello world\n";  
» Hello world  
  
> print 'Hello world\n';  
» Hello world\n
```

Perl syntax:
Numbers don't need quotes

```
> print 42;  
  
» 42
```

Perl syntax:
Parentheses are optional

```
> print("Hello world");  
» Hello world  
  
> print "Hello world";  
» Hello world
```

Perl regular expressions:
Simple matching on \$_

```
> while (<>) {  
    if (/foo/) {  
        print;  
    }  
}
```

Perl regular expressions:
Simple matching-scalars

```
> if ($a =~ /foo/) { ... }  
  
> if ($a =~ /foo/i) { ... }
```

Perl regular expressions:
Simple substitution on \$_

```
> while (<>) {  
    s/foo/bar/;  
    print;  
}
```

Perl regular expressions:
Simple substitution

```
> $a =~ s/foo/bar/;  
  
> $a =~ s/foo/bar/g;  
  
> $a =~ s/foo/sqrt(2)/ge;
```

Perl regular expressions:
Special patterns

```
> . a single character  
> \s whitespace character  
> \S non-whitespace character  
> \d a digit (0-9)  
> \D a non-digit  
> \w word character A-Z a-z 0-9 _  
> \W a non-word character
```

Perl regular expressions:
Special patterns

```
> [aeiou] single char in the set  
> [^aeiou] single char outside  
> (foo|bar|baz) foo or bar or baz  
> ^ start of string  
> $ end of string  
  
> /^[aeiou](l|k|n)[aeiou]$/
```

Perl regular expressions:
Quantifiers

```
> * zero or more  
> + one or more  
> ? zero or one  
> {3} exactly 3  
> {3,6} between 3 and 6  
> {3,} 3 or more
```

Perl regular expressions:
Examples

```
> /\d+/  
> /\s*$/  
> /^$/  
> /(\d\s){3}/  
> /(a.)+/
```

Perl regular expressions:
Capturing with parens

```
> if ($email =~ /(.)@(.)/) {  
    print "Username is $1\n";  
    print "Hostname is $2\n";  
}
```

Perl variables:
Three types

- Scalars: \$foo
- Arrays: @foo
- Hashes: %foo

Perl variables:
Scalars-string or number

```
> my $animal = "camel";  
> my $answer = 42;  
> print $animal;  
  » camel  
> print $answer;  
  » 42
```

Perl variables:
Scalars-interpolation

```
> print "The animal is $animal\n";  
  » The animal is camel  
> print "Square of $answer is ",  
  $answer * $answer, "\n";  
  » Square of 42 is 1764
```

(Interpolation only works with double quotes, not single quotes)

Perl variables:
Scalars-special (\$_)

```
> print;  
  » prints the contents of $_  
> length;  
  » returns the length of $_
```

Perl variables:
Arrays-zero indexed

```
> my @animals = ("camel", "owl");  
> print $animals[0];  
  » camel  
> print $animals[1];  
  » owl
```

Perl variables:
Arrays-number of elements

```
> my @mixed = ("camel", 42, 3.14);  
> print $mixed[$#mixed];  
  » 3.14  
> if (@mixed == 3) { print "foo"; }  
  » foo
```

Perl variables:
Arrays-slicing

```
> my @numbers = (10, 11, 12, 13);  
> @numbers[0, 1];  
  » (10, 11)  
> @numbers[0..2];  
  » (10, 11, 12)  
> @numbers[1..$#numbers];  
  » (11, 12, 13)
```

Perl variables:
Arrays-sort, reverse

```
> my @a = ("rat", "bat", "cat");  
> my @sorted = sort @a;  
  » ("bat", "cat", "rat")  
> my @backwards = reverse @a;  
  » ("cat", "bat", "rat")
```

Perl variables:
Arrays-special (@_, @ARGV)

```
> @ARGV  
  » the command line arguments to your  
  script  
> @_  
  » the arguments passed to a subroutine
```

Perl variables:
Hashes-key value pairs

```
> my %fruit_color = (  
  "apple", "red",  
  "banana", "yellow"  
);  
> $fruit_color{"apple"}  
  » "red"
```

Perl variables:
Hashes-alternative syntax

```
> my %fruit_color = (  
  apple => "red",  
  banana => "yellow",  
);  
> $fruit_color{"banana"}  
  » "yellow"
```

Perl variables:
Hashes-keys and values

```
> my @fruits = keys %fruit_color;  
  » ("apple", "banana")  
> my @colors = values %fruit_color;  
  » ("red", "yellow")
```

Perl variables:
Hashes-special (%ENV)

```
> $ENV{USER}
  » dyuret
> $ENV{HOST}
  » dal i . eng. ku. edu. tr
```

Perl variables:
References-scalar pointers

```
> my @animals=("rat", "bat", "cat");
> my $aref = \@animals;
> $aref->[0];
  » "rat"
> my $href = \%fruit_colors;
> $href->{apple}
  » "red"
```

Perl variables:
References-declaring

```
> my $aref = [10, 11, 12, 13];
> $aref->[2];
  » 12
> my $href = { banana => "yellow",
              apple => "red" };
> $href->{apple}
  » "red"
```

Perl variables:
References-embedding

```
> my $a = [10, 11, 12, 13];
> my $b = { banana => "yellow",
           apple => "red" };
> my $c = [ $a, $b ];
> $c->[0][2]
  » 12
> $c->[1]{banana}
  » "yellow"
```

Perl subroutines:
Simple example

```
> sub log {
    my $logmessage = shift;
    print LOGFILE $logmessage;
}
```

Perl subroutines:
The argument array @_

```
> my $logmessage = shift;
> my ($msg, $priority) = @_;
> my $msg = $_[0];
```

Perl subroutines:
Returning values

```
> sub square {  
    my $num = shift;  
    my $result = $num * $num;  
    return $result;  
}
```

Perl subroutines:
Variable scope

```
> my $var = "value";  
> $var = "value";
```

Perl subroutines:
Variable scope

```
> my $a = "foo";  
if ($some_condition) {  
    my $b = "bar";  
    print $a;  
    print $b;  
}  
print $a;  
print $b;
```

Perl magic

```
> Printing integers from 1 to 100  
  
    print "$_\n" for 1 .. 100;
```

Perl magic

```
> Printing letters from a to z  
  
    print "$_\n" for 'a' .. 'z';
```

Perl magic

```
> Computing the intersection of  
two arrays  
  
$mark{$_}++ for @array1;  
@ans = grep($mark{$_}, @array2);
```

Perl magic

> Shuffling an array randomly

```
@a = 1 .. 30;
push(@b, splice(@a, rand(@a), 1))
while @a;
print @b;
```

Perl magic

> Selecting a random line from a file

```
while(<>) {
    rand($.) < 1 && ($it = $_);
}
print $it;
```

Perl magic

> Printing lines after a pattern match

```
perl -ne 'print if
(/caravan/ ? ($c=5) : (--$c > 0))'
```

Perl magic

> Extracting selected columns by character

```
perl -lne 'print substr($_, 3, 5)'
```

Perl magic

> Extracting selected columns by word

```
perl -lane 'print $F[2];'
```

Perl magic

> Printing lines in sorted order

```
perl -e 'print sort <>'
```

Perl magic

> Printing lines in reverse order

```
perl -e 'print reverse <>'
```

Perl magic

> Reversing lines by characters

```
perl -lne 'print reverse($_)'
```

Perl magic

> Reversing lines by words

```
perl -lane  
'print join(" ", reverse @F)'
```

Perl magic

> Printing a file with multiple blank lines squeezed into one

```
perl -ne  
'print if /\S/ || !$s;  
$s = /\s*$/;'
```

Perl magic

> Putting commas into integers

```
sub comma {  
  my $n = shift;  
  1 while $n =~ s/(.*\d)(\d\d\d)/\1, \2/;  
  $n;  
}
```

Perl magic

> Counting word frequencies

```
perl -ane '  
$cnt{$_}++ for @F;  
END { print "$cnt{$_}\t$_\n"  
      for keys %cnt }'
```

Where to go from here

> Useful perl man pages

- » perl intro introduction
- » perl func functions
- » perl run command line options
- » perl var special variables
- » perl op operators
- » perl re regular expressions