COMP284 Scripting Languages

Lecture 7: Perl (Part 6)

Handouts

Department of Computer
School of Electrical Engineering, Electronics, an
University of Liverpo

Contents

I/O Connections

- Perl programs interact with their environment via I/O connections
- A filehandle is the name in a Perl program for such an I/O connection, given by a Perl identifier
  Beware: Despite the terminology, no files might be involved
- There a

STD	
STD	
STDERR	Standard Error, for error output, typically defaults to the terminal
DATA	Input from data stored after Perl program
ARGV	Iterates over command-line filenames in @ARGV
ARGVOUT	Points to the currently open output file when doing edit-in-place processing with -i `perl -pi -e 's/cat/dog/' file`

I/O Connections

Except for the six predefined I/O connections, all other I/O connections

- need to be opened before they can be used
  `open filehandle, mode, expr`
- should be closed once no longer needed
  `clos`
- can be u
  `<filehandle>`
- can be used to write to
  `print filehandle list`
  `printf filehandle list`
- can be selected as default output
  `select filehandle`

I/O Connections

Example:

```
open INPUT, "<", "oldtext.txt" or die "Cannot␣open␣file";
open OUTPUT, ">", "newtext.txt";
while (<INPUT>) {
   s!(\d+) degrees Fahrenheit!
     sp
   print O[illegible]
}
close(INP[illegible]
close(OUTPUT);
```

`oldtext.txt`:

```
105 degrees Fahrenheit is quite warm
```

`newtext.txt`:

```
41 degrees Celcius is quite warm
```

Opening a filehandle

```
open filehandle, expr
open filehandle, mode, expr
```

- Opens an I/O connection specified by *mode* and *expr* and associates it with *filehandle*
- *expr*
- *mode*

Mod			
<	read file		
>	write file	yes	
>>	append file	yes	
+<	read/write file		
+>	read/write file	yes	yes
+>>	read/append file	yes	
|-	write to command	yes	
-!	read from command	yes	

Closing a filehandle

`close`

`close` *filehandle*

- Flushes the I/O buffer and closes the I/O connection associated with *filehandle*
- Retur
- Close

Reading

<*filehandle*>

- In a scalar context, returns a string consisting of all characters from *filehandle* up to the next occurrence of $/ (the input record separator)
- In a list co
  of *fi*
  (Defa

```perl
open INPUT, "<", "oldtext.txt" or die "Cannot␣open␣file";
$first_line = <INPUT>;
while ($other_line = <INPUT>) { ... }
close INPUT;

open LS, "-|", "ls␣-1";
@files = <LS>;
close LS;
foreach $file (@files) { ... }
```

Selecting a filehandle as default output

`select`

`select` *`filehandle`*

- If *filehandle* is supplied, sets the new current default filehandle for output
  - ⇝ wr dle
  - ⇝ Re ehandle
- Retur

Printing

```
print filehandle list
print filehandle
print list
print
```

- Print a s
- If *fil*
- If *lis*
- The current value of `$,` (if any) is printed bet em (Default: `undef`)
- The current value of `$\` (if any) is printed aft has been printed (Default: `undef`)

Printing: Formatting

`sprintf(`*format*`, `*list*`)`

- Returns a string formatted by the usual printf conventions of the C library function sprintf (but does not by itself print anything)

```
sprintf "(%10.3f)" 1234.5678
```

forma
and put

```
(   1234.568)
```

See `http://perldoc.perl.org/func` r
further details

Printing: Formatting

```
printf filehandle format, list
printf format, list
```

- Equivalent to
  `print filehandle sprintf(format, list)`
  excep

Printing: Formatting

Format strings can be stored in variables and can be constructed on-the-fly:

```
@list = qw(wilma dino pebbles);
$format = "The␣items␣are:\n". ("%10s\n" x @list);
printf $format, @list;
```

Output:

```
The items are:
     wilma
      dino
   pebbles
```

(The code above uses the 'quote word' function
to generate a list of words.
See http://perlmeme.org/howtos/perlfunc/qw_function.html
for details)

Here documents

- A here document is a way of specifying multi-line strings in a scripting or programming language
- The basic syntax is

```
<<identifier
here docu
identi
```

- *ide*
  here document ends
- *identifier* might optionally be surrounded by
  or backticks
  An unquoted identifier works like a double-quoted o
- The here document starts on the following line
- The terminating string *identifier* must appear by itself (unquoted and with no surrounding whitespace) after the last line of the here document

Here documents: Double-quotes

```
$title = "My␣HTML␣document"
print <<"END";
Content-type: text/html

<!DOCTYPE html>
<HTML>
<HEADER><TITLE>$title</TITLE></HEADER>
<BODY>
  <H1>$tit[illegible]
  Lots of HTML mar[illegible]
</BODY>
</HTML>
END
```

```
Content-type: text/html

<!DOCTYPE html>
<HTML>
<HEADER><TITLE>My HTML document</TITLE></HEADER>
<BODY>
  <H1>My HTML document</H1>
  Lots of HTML markup here
</BODY>
</HTML>
```

The double-quotes in "END" indicate that everything between the opening "END" and [illegible] should be [illegible]ted [illegible]

Here documents: Single-quotes

```
$title = "My␣HTML␣document"
print <<'END';
Content-type: text/html

<!DOCTYPE html>
<HTML><HE
<BODY></B
END
```

The singl END' and
END should be treated like a single-quoted string

⤳ no variable interpolation is applied

⤳ $title will not be expanded

```
Content-type: text/html

<!DOCTYPE html>
<HTML><HEADER><TITLE>$title</TITLE></HEADER>
<BODY></BODY></HTML>
END
```

Here documents: Backticks

```
$command = "ls";
print <<`END`;
$command -1
END
```

The backticks in `END` tell Perl to run the here document as a shell script (with the h

```
handouts.
handouts.
handouts.pdf
handouts.tex
```

Here documents: Variables

Here documents can be assigned to variables and manipulated using string operations

```
$header = <<"HEADER";
Content-type: text/html

<!DOCTYPE ht
<HTML><HE
HEADER

$body = <<"BODY";
<BODY>
  <H1>$title</H1>
  Lots of HTML markup here
</BODY>
</HTML>
BODY

$html = $header.$body;
print $html;
```

Invocation Arguments

- Another way to provide input to a Perl program are invocation arguments (command-line arguments)

```
./perl_program arg1 arg2 arg3
```

- The invocation arguments given to a Perl program are stored in the specia

perl

```
print "Nu
for ($index=0; $index <= $#ARGV; $index++) {
  print "Argument␣$index:␣$ARGV[$index],"\n";
}
```

```
./perl_program1 ada 'bob' 2
```

Output:

```
Number of arguments: 3
Argument 0: ada
Argument 1: bob
Argument 2: 2
```

Options

- There are various Perl modules that make it easier to process command-line options
  --scale=5 --debug --file='image.png'

- One such module is Getopt::Long:
  http

- The m

- GetO
  @ARGV according to an option specification
- Arguments that do not fit to the option specificatio ARGV

- GetOptions returns true if @ARGV can be

Options: Example

perl_program2:

```
use Getopt::Long;
my $file = "photo.jpg";
my $scale = 2;
my $debug = 0;

$result = GetO




print "Debug:␣$debug;␣Scale:␣$scale;␣File:␣$file\n";
print "Number␣of␣arguments:␣",$#ARGV+1,"\n";
print "Arguments:␣",join(",",@ARGV),"\n";
```

```
./perl_program2 --scale=5 --file='image.png' arg1 arg2
```

```
Debug: 0; Scale: 5; File: image.png
Number of arguments: 2
Arguments: arg1, arg2
```

Revision

Read

- Chapter 5: Input and Output

of

R. L. Sch

Learni

O'Reilly, 2011.

- `http://perldoc.perl.org/perlop.`
- `http://perldoc.perl.org/perlop.html#Quote-Like-Operators`
- `http://perldoc.perl.org/Getopt/Long.html`