COMP284 Scripting Languages
Lecture 7: Perl (Part 6)
Handouts

Ulrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

1. Input/Output
   - Filehandles
   - Open
   - Close
   - Read
   - Select
   - Print
   - Here documents
2. Arguments and Options
   - Invocation Arguments
   - Options

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 are six pre-defined filehandles

`STDIN`	Standard Input, for user input, typically the keyboard
`STDOUT`	Standard Output, for user output, typically the terminal
`STDERR`	Standard Error, for error output, typically defaults to the terminal
`DATA`	Input from data stored after `__END__` at the end of a 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
  `close` *filehandle*
- can be used to read from
  <*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!
     sprintf("%d",(($1-32)*5/9)+0.5)." degrees Celsius"!e;
   print OUTPUT;
}
close(INPUT);
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* specifies a file or command
- *mode* is one of the following

Mode	Operation	Create	Truncate
<	read file		
>	write file	yes	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*
- Returns true if those operations succeed
- Closes the currently selected filehandle if the argument is omitted

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 context, returns a list of strings representing the whole content of *filehandle* separated into string using $/ as a separator (Default value of $/: newline \n)


```
1 open INPUT, "<", "oldtext.txt" or die "Cannot open file";
2 $first_line = <INPUT>;
3 while ($other_line = <INPUT>) { ... }
4 close INPUT;
5
6 open LS, "-|", "ls -1";
7 @files = <LS>;
8 close LS;
9 foreach $file (@files) { ... }
```

Selecting a filehandle as default output

```
select
select filehandle
```

- If *filehandle* is supplied, sets the new current default filehandle for output
  - ⇝ `write` or `print` without a filehandle default to *filehandle*
  - ⇝ References to variables related to output will refer to *filehandle*
- Returns the currently selected filehandle

Printing

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


- Print a string or a list of strings to *filehandle*
- If *filehandle* is omitted, prints to the last selected filehandle
- If *list* is omitted, prints `$_`
- The current value of `$,` (if any) is printed between each *list* item (Default: `undef`)
- The current value of `$\` (if any) is printed after the entire *list* 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
```

format a floating-point number with minimum width 10 and precision 3 and put the result in parentheses:

```
(  1234.568)
```

See `http://perldoc.perl.org/functions/sprintf.html` for further details

Printing: Formatting

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

- Equivalent to

  ```
  print filehandle sprintf(format, list)
  ```

  except that $\ (the output record separator) is not appended

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 `qw()`
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 document
identifier
```


- *identifier* declares the terminating string that will indicate where the here document ends
- *identifier* might optionally be surrounded by double-quotes, single-quotes or backticks
  An unquoted identifier works like a double-quoted one
- 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>$title</TITLE></HEADER>
<BODY>
  <H1>$title</H1>
  Lots of HTML markup here
</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 the closing END should be treated like a double-quoted string

Here documents: Single-quotes

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

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


The single-quotes in 'END' indicate that everything between '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 here document treated like a double-quoted string)

```
handouts.aux
handouts.log
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 html>
<HTML><HEADER><TITLE>$title</TITLE></HEADER>
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 special array @ARGV

perl_program1:


```
print "Number of arguments: ",$#ARGV+1,"\n";
for ($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://perldoc.perl.org/Getopt/Long.html`
- The module provides the `GetOptions` function
- `GetOptions` parses the command line arguments that are present in `@ARGV` according to an option specification
- Arguments that do not fit to the option specification remain in `@ARGV`
- `GetOptions` returns true if `@ARGV` can be processed successfully

Options: Example

perl_program2:

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

$result = GetOptions ("debug"   => \$debug,   # flag
                      "scale=i" => \$scale,   # numeric
                      "file=s"  => \$file);   # string

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. Schwartz, brian d foy, T. Phoenix:
Learning Perl.
O'Reilly, 2011.

- `http://perldoc.perl.org/perlop.html#I%2fO-Operators`
- `http://perldoc.perl.org/perlop.html#Quote-Like-Operators`
- `http://perldoc.perl.org/Getopt/Long.html`