



G64HLL

High Level Languages

Lecture 11- PERL

Dr. Natalio Krasnogor

Natalio.Krasnogor@Nottingham.ac.uk



The Perl Section

- 7 lectures (probably)
 - Intro to Perl, references, scalars, conditionals
 - Arrays, hashes, loops, print
 - Getting input, files, sub-routines, scoping
 - Regular expressions
 - Additional material(?)



Today's lecture

- Intro to Perl
- Hello World
- Running Perl programs
- Scalars
 - Numerical operations
 - String operations
- Conditionals



What is Perl?

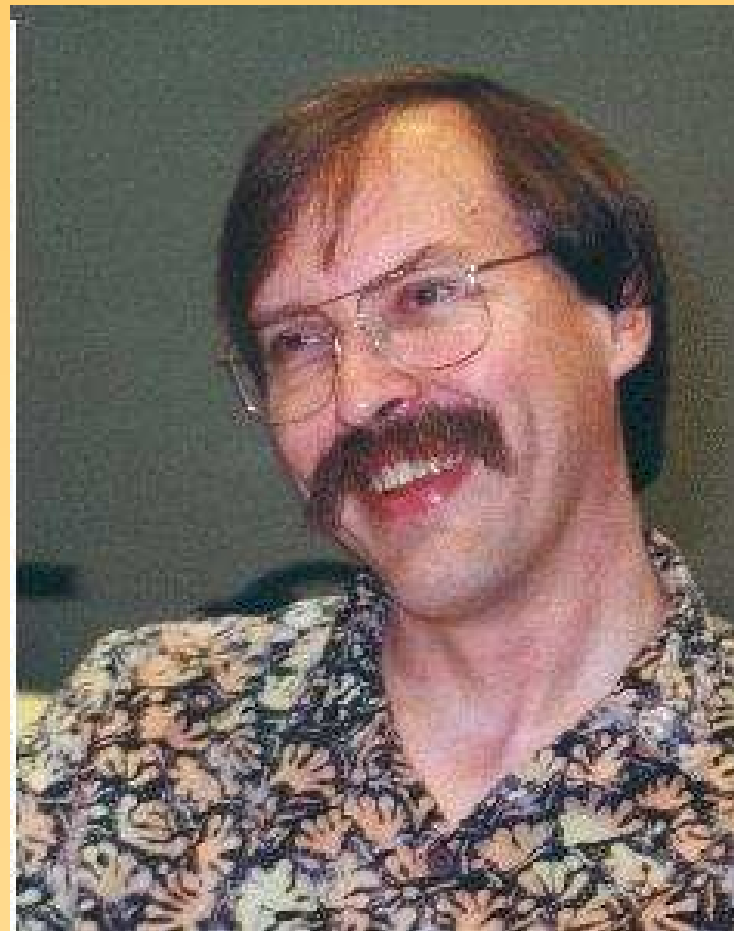
- Known as a scripting language
- Devised by Larry Wall back in 1987
- Modern version took shape around 1994

G64HLL: High-Level Languages – Website:
<http://www.cs.nott.ac.uk/~nxk/TEACHING/G6DHLL/main.html>



The University of
Nottingham

Larry Wall





What is Perl good at?

- Quick program development
- Low hassle
- Text processing
- Task automation
- Web applications
- Glue



What is Perl bad at?

- Speed
 - Comparable to Java, slower than Pascal/C++
- Graphical interfaces
 - Possible, but not recommended
- System-level programming



Getting Perl

- Comes installed with most versions of UNIX, BSD and Linux
- Good Windows package from <http://www.activestate.com>



References – Books

- Perl and CGI for the World Wide Web
 - Elizabeth Castro
 - Peachpit press
 - £15
- Last year's course text
- Focussed very much on Web side of Perl
- Uses different Web library to this course, but still some good information



References – Books

- Programming Perl (3rd edition)
 - Larry Wall, Tom Christiansen and Jon Orwant
 - O'Reilly
 - £29
- The daddy of all Perl books
- Way more information than you need for this course



References – Books

- Learning Perl (3rd edition)
 - Randal L. Swartz and Tom Phoenix
 - O'Reilly
 - £20
- Expanded version of the start of Programming Perl



References – Web

- <http://www.perl.com>
 - The official Perl site

- <http://www.perl.com/pub/q/FAQs>
 - **VERY USEFUL**



References – Other

- `man perl` under UNIX
- HTML documents with ActivePerl



Hello World!

- Hello World in Pascal:

```
program hello;  
begin  
    writeln('Hello World');  
end.
```



Hello World!

- Hello World in Java:

```
class hello {  
    public static void main(int args, char[] argv) {  
        System.out.println("Hello World");  
    }  
}
```



Hello World!

- Hello World in Perl:

```
print "Hello World";
```



Creating a Perl program

- Like Java, but unlike Delphi, you do not use any particular development environment
- Any text editor will do
- ...but some make it easier than others
- A program that can do syntax highlighting is a good idea (emacs my choice!)



A choice of editing programs

- Windows
 - Few that are any cop
 - Editpad Pro (<http://www.jgsoft.com>) is probably the best, but it's not free (\$30)
- UNIX
 - emacs has some Perl support
 - NEdit (<http://www.nedit.org>) is excellent and free



But it's up to you...

- NEdit is also a good environment to develop in
- Developing directly in UNIX makes more sense! Don't fear the UNIX!
- If you *really* want to use a Windows editor: don't just save to your H: drive, use FTP in ASCII mode instead



Hello World revisited

- To run Hello World type the program into NEdit
- Save as hello.pl
- In the UNIX shell type:

```
nxk$ perl hello.pl
```

(NB: `nxk$` is the UNIX prompt, don't type that bit)

- The output will be:

```
Hello Worldnxk$
```



Hello World revisited

- Perl doesn't automatically add a new-line to the end of `print` statements
- Instead the `\n` character should be used wherever you want a newline, e.g.:

```
print "Hello World\n";
```

- This would give:

```
nxk$ perl hello.pl
```

```
Hello World
```

```
nxk$
```



Hello World revisited

- You can use `\n` anywhere within a print statement, e.g.:

```
print "Hello\n\nWorld\n";
```

to give:

```
nxk$ perl hello.pl
```

```
Hello
```

```
World
```

```
nxk$
```



Escape characters

- `\n` is an example of an “escape character”
- The backslash tells you it’s an escape
- The `n` is this particular character (new-line)
- You may also need to escape some other characters if you want them to print normally instead of having a special meaning, e.g.:
 - double quotes
 - backslash



Comments

- Comments in Perl are flagged with the # symbol
- From the # to the end of the line will be treated as a comment
- e.g.:

```
# a simple program
```

```
print "Hello World\n"; # print to screen
```



Running your program

- Using `perl filename` is fine
- It is possible to avoid having to type `perl` and make it so just typing *filename* will run the program
- Since Perl files are just text files you need to tell UNIX what program is used to interpret them
- The standard UNIX way of doing this is by having a `#!` (pronounced “she-bang”) line as the first line in your program



Running your program

- The `#!` Line for Perl on the CS machines is:

```
#! /usr/bin/perl
```

- You'll also need to tell UNIX that your file is executable, by typing an appropriate `chmod` line into the UNIX shell:

```
chmod 755 hello.pl
```

- If just typing the filename doesn't run the program, try `./filename`



Hello World – Final version

- Hello World now looks like:

```
#! /usr/bin/perl
```

```
# Hello World - By your name here
```

```
# version 1.0
```

```
# print a message on the screen
```

```
print "Hello World!\n";
```



Scalars

- Simple variables are known as ‘scalars’
- Identified by using a leading dollar symbol (sometimes called the ‘string’ symbol)
- No need to declare type before use
- This is a valid Perl program:

```
$age = 24;           # integer
$name = "duncan";   # string
$pi = 3.14;         # real/float
print $name, " is ", $age, " pi is ", $pi, "\n";
```



Scalars – Type conversion

- Conversion between types is automagic:

```
$age      = 24;  
$pi       = 3.14;  
$age      = $age + $pi;  
  
# $age is now a real  
# with a value of 27.14
```



Scalars – Type conversion

- Conversion between types is *really* automagic:

```
$street = "Acacia Road";      # a string
$num    = 29;                 # an integer
```

```
# we use a dot to join strings
```

```
# integers/reals can be treated
```

```
# as strings for this
```

```
# $erics_address becomes "29 Acacia Road"
```

```
$erics_address = $num . " " . $street;
```



Scalars – Type conversion

```
# if strings look like numbers they  
# can be used as numbers
```

```
$number    = "12345";           # string
```

```
$total     = $number + 4;
```

```
# $total is 12349 (integer)
```



Scalars – Numerical functions

- Addition

```
$a = 1 + 1;           # sets $a to 2
```

```
$a += 4;             # adds 4 to $a
```

```
$a++;                # adds 1 to  
$a
```

- Subtraction

```
$a = 10 - 5;         # sets $a to 5
```

```
$a -= 6;             # subtracts 6 from $a
```

```
$a--;                # subtracts 1 from $a
```



Scalars – Numerical functions

- Multiplication

```
$a = 2 * 3;           # sets $a to 6  
$a *= 10;           # multiplies $a by 10
```

- Division

```
$a = 10 / 3;         # sets $a to 3.3333  
$a /= 2;            # halves $a
```

- Modulus

```
$a = 10 % 3;        # sets $a to 1  
$a %= 2;            # sets $a to modulus 2  
# of itself
```



Scalars – String functions

- *substr string, pos, [num, replacement]*
 - Returns the section of *string* starting at *pos*
 - If *num* is specified only *num* characters are returned
 - If *replacement* is specified, the section is replaced with the new value



Scalars – String functions

```
$line = "dogs chase cats";
```

```
# set $victim to "cats"
```

```
$victim = substr $line,11;
```

```
# set action to "chase"
```

```
$action = substr $line, 5, 5;
```

```
# set $line to "dogs fear cats"
```

```
substr $line, 5, 5, "fear";
```



Scalars – String functions

- If you specify a negative number for *pos*, it counts from the end of the string not the start

```
$line = "cows eat grass";
```

```
$food = substr $line, -5;      # grass
```

```
$action = substr $line, -9, 3; # eat
```



Scalars – String functions

- **x**
 - The **x** operator repeats a string

```
$laugh = "ha" x 3;           # "hahaha";
```

```
print "-" x 80;           # row of dashes
```



Scalars – String functions

- ***chop string***

- Removes the last character from string

- ***chomp string***

- Removes the last character from string, if the last character was a carriage return



Scalars – String functions

```
$line = "Lilly the Pink";  
chop $line;  
print $line;                # Lilly the Pin  
  
# chop returns the letter removed  
$letter = chop $line;  
print $letter;              # n
```



Scalars – String functions

- `length $string`
 - Returns length of a string

```
$class = "asleep";  
print length $class;           # 6
```



Other data types

- As well as scalars, Perl also has
 - Arrays
 - Hashes

- We'll come back to those in a bit



If

- `if` statements are pretty much as other languages:

```
if (some test) {  
    # code here  
} elseif (some other test) {  
    # code here  
} else {  
    # if neither test met, do this  
}
```



If – Tests

- A test is anything which evaluates to True or False
- For example $a > b$ will be true if a is greater than b
- You can join tests with $\&\&$ (and) and $||$ (or)



If – Numerical tests

==	Equals
>	Greater than
<	Less than
>=	Greater or equal to
<=	Less or equal to
!=	Not equal



If – String tests

<code>eq</code>	Equals
<code>ne</code>	Not-equals

- There are others, but you rarely need them



If – Examples

```
$a = 10;  
if ($a > 5) {  
    print "a is big enough\n";  
}
```

```
if ( ($a > 5) && ($a <= 10) ) {  
    print "a is in range\n";  
}
```



If - Negation

- You can use ! to negate any test:

```
if ( !($a > 10) ) {  
    print "a isn't bigger than 10\n";  
}
```



If - Truth

- You don't always need to use a comparison operator to get a truth value
- Any number other than 0 is true
- Any string other than "0" and the empty string are true



If - More examples

```
# both of these will make it into the 'if'
$a = 5;
if ($a) {
    print "a is true\n";
}

$b = "Hello";
if ($b) {
    print "b is true\n";
}
```



If - Doing it backwards

- Perl supports putting the `if` statement after the command, e.g.:

```
print "a is big\n" if ($a > 50);
```



Unless

- `unless` is the logical opposite of `if`

```
unless ($a > 50) {  
    print "a is small\n";  
} else {  
    print "a is big\n";  
}
```



Unless - Backwards

- Like `if`, `unless` can also be used after the command

```
print "a is small" unless ($a > 50);
```



Lecture 1 – Summary

- That's all folks, we covered:
 - Intro to Perl
 - Hello World
 - Running Perl programs
 - Scalars
 - Numerical operations
 - String operations
 - Conditionals

LOONEY TUNES

That's all Folks!

A WARNER BROS. CARTOON

DRESSED BY DESIGN © 1968 WARNER BROS. ENTERTAINMENT CO.

MUSIC BY BOB WATNER BROS. © 1968 WARNER BROS.

ALL RIGHTS AND CHARACTER TRADEMARKS OF WARNER BROS.