

CS306: Introduction to Perl

Intro and Section#1: What is Perl? and Scalars

U. of Alabama at Birmingham
Dept. of Computer & Information Sciences

Slide 1

Intro: What is Perl?

Introduction
History
Philosophy
Hello World

Slide 2

What is Perl?

- A multi-purpose, cross-platform interpreted scripting language that fits somewhere between shell scripting and languages like C or Java. *
- A freely available open-source project
- “Practical Extraction and Report Language” or “Pathologically Eclectic Rubbish Lister”

* This, like a lot of Perl things, isn't strictly true, but if I explained all of Perl's exceptions, this course would be Introduction to Perl's Exceptions instead.

Slide 3

What do you use Perl for?

- A question with a shorter answer might be “When can't you use Perl?”
- Perl is good at system administration tasks, text parsing, database access, networking, graphical programming, web scripting, quick prototyping
- Used by sysadmins, web scripters, bioinformaticists, physicists, mathematicians, etc...

Slide 4

Perl's Strengths

- Everything from quick and dirty scripts to massive programs with multiple programmers working for many months or years
- It has been said that Perl is optimized for “problems which are about 90% text and 10% everything else” [*from Learning Perl*]
- Really a jack of all trades, and master of some

Slide 5

History of Perl

- Created by Larry Wall. Perl 1.0 in 1987.
- He started by gluing together UNIX utilities that he found useful
- Many people over the years have contributed to the growth of Perl through modules
- Current version is 5.8.6

Slide 6

Philosophy of Perl

- TIMTOWTDI – There Is More Than One Way To Do It
- Perl generally does not have One Right And True Way to do something. This is a double-edged sword.
 - It's very flexible, and...
 - ...you have to be careful not to abuse that fact.

Slide 7

Philosophy of Perl 2

- Perl will not stop you if you want to:
 - Write your entire program on one line
 - Use your integer like a string
 - Use a string like an integer
 - Assign a value to the 17th element of an array that only has 3 elements at the moment
- Perl will not hold your hand, but it's so easy to use, you won't need it anyway

Slide 8

Philosophy of Perl 3

- Perl loves shortcuts
 - Perl has shortcuts for **everything**. You'll learn to love this eventually, but it will be maddening at first. This is because Larry had a choice between...
- Easy to learn vs. easy to use
 - and he chose easy to use

Slide 9

Philosophy of Perl 4

- Perl will often guess (usually correctly) at what you were trying to do
 - For example, if Perl sees you try to assign an array to a scalar, it will guess that what you really wanted to know was the length of the array
- No hard limits
 - Perl tries not to impose arbitrary limits like variable name length or size of a string

Slide 10

Perl at CIS and at Home

- In CIS, Perl is available on the blazer machines. You will need a CIS UNIX account to use these.
- It may also be available on the Windows machines *
- Perl, being free and open-source, is available for you to install at home or on your laptop

* Insert UNIX Sales Pitch here

Slide 11

How to get Perl

- For UNIX-like OS (Linux, OS X, etc...) - You already have Perl. Popular UNIX-like OS flavors have been shipping with Perl for many years.
- For Windows, ActivePerl – www.activestate.com/activeperl *
- Can also build from source – get the source from the CPAN (CPAN.org) or perl.com

* What, my sales pitch didn't work?

Slide 12

What is the CPAN?

- Comprehensive Perl Archive Network at cpan.org
- One-stop shopping for Perl source code, documentation, and extensions
- Most of the class will focus on Perl's core functionality, but towards the end of the semester we will explore CPAN a bit more

Slide 13

A first program - Hello World

```
#!/usr/bin/perl  
# A first Perl program  
print "Hello World!\n";
```

Slide 14

A first program - Hello World

```
#!/usr/bin/perl  
# A first Perl program  
print "Hello World!\n";
```

sh-bang or *shebang* line

A comment

A typical Perl statement

Slide 15

Run The Program

- # chmod a+x helloworld
- # ./helloworld
Hello World!
#

Slide 16

Run The Program

- # chmod a+x helloworld
- # ./helloworld
Hello World! Make the program executable
#

Run the program

Slide 17

Lifecycle of a Perl Program

- When we typed ./helloworld a lot happened
 - The shell read the shebang line and went out and loaded the perl binary (if not loaded already)
 - The perl binary in turn parsed through the text of the program and used its internal compiler to produce *bytecode*
 - The bytecode was executed, producing the output on the terminal, and the program exited

Slide 18

Section 1: Scalars

What are scalars?
Creating scalar variables
Operating on scalars

Slide 19

What are scalars?

- In Perl, a *scalar* is a singular piece of data, either a number or a string
- Some examples:
 - 14
 - hello
 - 5.64234
 - This sentence is a scalar!
 - <the entire text of the Programming Perl book>

Slide 20

What are scalars? 2

- Perl treats numbers and strings nearly interchangeably
- If you use a string where Perl expects a number, it will use the string as a number, and vice versa. Examples later.

Slide 21

Numbers

- Integers
5
745634
- Floating Point
5.2304
3.88e12
-56.3421

Slide 22

Numbers in Various Bases

- Decimal
46
- Hexadecimal
0x2e
- Binary
0b101110
- Octal
056 # Just a leading zero in this case

Slide 23

Strings

- String *literals** are represented in a Perl program within quotes
'hello!'
'Four score and seven years ago'
'What\'s for dinner?'
- Everything within the single quotes is interpreted literally. Use `\` to represent `'` in your strings.

* A literal is a piece of data represented directly in your program as opposed to being stored inside a variable. i.e. "Hard-coded".

Slide 24

Double-Quoted Strings

- To represent *control characters* within strings, use double-quotes instead.
“Go Blazers!\n”
“Fran\tFabrizio”
“She said, \”Take me to your leader.\””
- There are lots of control characters and the \ takes on a lot of power within double quotes. See *Programming Perl* page 61.

Slide 25

Numbers <--> Strings

- Perl will convert strings to numbers when needed
'35' + 4 # Equals 39
“16fran34” / “ 8” # Equals 2
“tony” + 3 # Equals 3, non-numeric string converts to 0
- Numbers to strings works as well
“Y” . 2 . “K” # Equals “Y2K”

Slide 26

Operating on Scalars

- For numbers, all the usual *operators*
+ - * / % **
- For strings, we have
 - The concatenation operator
“base” . “ball” # Equals “baseball”
 - The repetition operator
'ho' x 3 # 'hohoho'
3 x 3 # “333” Don't get fooled :-)

Slide 27

Scalar Variables

- Perl uses the \$ to indicate a scalar *variable*
\$name
\$return_value
\$aLengthyVariableName
- Variable names are case-sensitive, can contain alphanumeric and underscore characters, cannot start with a digit, and can be of any length

Slide 28

Scalar Assignment

- `$day = 'Wednesday';`
- `$grade = '95';`
- `$bonus = 5;`
- `$total = $grade + $bonus; # 100`
- `$extra_credit = 10;`
- `$total += $extra_credit; # 110`
- `$day .= ' afternoon'; # "Wednesday afternoon"`

Slide 29

String Interpolation

- When using double-quotes, not only does the `\` become more powerful, but you can also perform *variable substitution*
`$day = "Wednesday";`
`$phrase = "$day night"; # Wednesday night`
`$literal = '$day night'; # $day night`

Slide 30

String Interpolation 2

- When perl sees a `$` inside double-quotes, it tries to form the longest variable name possible
`$word = 'dog';`
`$string = "The $word barks."; # The dog barks`
`$string = "Cats and $words"; # Cats and`

.... because `$words` is not defined. Instead:

```
$string = "Cats and ${word}s"; # Cats and dogs
```

Slide 31

String Interpolation 3

- Four ways to accomplish the same thing

```
$word = 'dog';  
$string = "Cats and ${word}s\n";  
$string = "Cats and $word" . "s\n";  
$string = "Cats and $word" . s . "\n";  
$string = "Cats and " . $word . "s\n";
```

Slide 32

Comparing Scalars

- For numbers, you can use:
`== != > < >= <=`
- For strings, you can use:
`eq ne gt lt ge le`
- A very common Perl mistake:
`'cat' == 'dog' # True, Perl makes both sides 0`
`'cat' eq 'dog' # False, what you meant to do`

Slide 33

undef

- If a scalar variable has not yet been assigned a value, it has the Perl special value *undef*
- `undef` behaves like 0 when used as a number and like an empty string when used as a string
`5 + $sum # equals 5`
`$sum++;`
`5 + $sum # equals 6`
`$newstring .= "hello"; # $newstring is "hello"`

Slide 34

Boolean Values

- Perl does not require or have a specific boolean data type
- The following evaluate to false
`undef 0 '0' ''`
- Everything else evaluates to true
- Use the `!` to get the opposite of a boolean
`(! $done) # if $done is 1, evaluates to 0`

Slide 35

Simple output

- `print "Hello World!\n";`
- `$weather = 'rainy';`
- `print "It will be $weather today.\n";`
- `print "My name is ";`
`print "Fran";`
`print "\n";`
- `print "My name is " . "Fran" . "\n";`

Slide 36

The if Construct

- `if (some condition) {`
 `# do this if condition is true`
`} else {`
 `# do this if condition is false`
`};`
- `if ($name eq 'Fran') {`
 `print "Hello Mr. Fabrizio\n";`
`}`

Slide 37

The if Construct 2

- `if (! $finished) { print "Keep working!\n"; }`
- `if ($pushups >= 30) {`
 `print "Great job!\n";`
`} elsif ($pushups >= 20) {`
 `print "Not too bad.\n";`
`} else {`
 `print "You'd better visit the new rec center.\n";`
`}`

Slide 38

Simple input

- Use `<STDIN>`
 - The full explanation of this will occur in a later lecture
- `print "Your name please: ";`
`$name = <STDIN>;`
`print "Hello $name!";`

Slide 39

Simple input 2

- Note that `<STDIN>` grabs an entire line of input from standard input (typically the keyboard), including the newline character at the end
- The `chomp()` function is handy here
 - `$name = <STDIN>;`
 `chomp $name;`
 - `chomp()` looks for and removes a trailing `\n`
 - Important to remember to remove it, otherwise it will mess up things like string comparisons

Slide 40

The End

- Grab the homework assignment on the way out.
- Post questions to the web forum
- Review today's material for the quiz next week.