1.0.1.8.1

Introduction to Perl
Session 1

- what is Perl?
- history of Perl
- writing and running a Perl script
- dealing with variables

what is Perl?

- Perl is a programming language (duh)
- Perl has a philosophy of pragmatism, creativity and fun
  - it lets you **get the job done**
  - it makes easy jobs easy and hard jobs possible
  - it makes it easy to manipulate numbers, text, files, strings, directories and processes
  - it's free, available on nearly every platform
  - **there's more than one way to do it**
  - it's simple to learn but deep enough to continue to stimulate for years to come
  - it's highly idiomatic – just like a language
  - it works on the **principle of least surprise**
  - it's remarkably extensible (Comprehensive Perl Archive Network)
- Perl is extremely rich – just like a language
  - you can pick up "conversational" Perl in a few weeks
  - you can write Perl poetry in a few months
  - you will speak Perl slang shortly after that

history of Perl

Larry Wall

- 1987 Perl 1.0 is released by Larry Wall
  - practical extraction and report language
  - pract
- 1988 Perl 2.0 is released
  - sort operator added, among other things
- 1989 Perl 3.0 is released
  - you can now pass things by reference, among other things
- 1991 Perl 4.0 is released
  - I'm reminded of the day my daughter came in, looked over my shoulder at some Perl 4 code, and said, "What is that, swearing?" *Larry Wall*
- 1995 Perl 5.0 is released
  - POD is introduced, among other things
- Perl 6.0 will be released when it's ready – current version 5.10.0 (13-may-08)
  - named parameters will be added, among other things

`http://history.perl.org/PerlTimeline.html`

`http://dev.perl.org/perl6/faq.html`

Myths of Perl

- Perl looks like line noise
- Perl is hard because ...
  - it has regexps
  - it has references
- Perl is just for UNIX
- Perl is just for one-liners – you can't build "real" programs with it
- Perl is just for the web
- Perl is too slow
- Perl is insecure

http://www.perl.com/pub/a/2000/01/10PerlMyths.html

a few notes before we begin

- Perl is a practical alternative to bashing your head against the wall
- anyone can learn Perl and make good use of it
- for every 1 hour learning Perl you will save 1 month of work
- train your eyes to quickly spot the difference between

{} [] ()

- whatever we are doing, thinking whether there is another way to do it
- Perl gives you a lot of freedom – control yourself!
- we won't write "Hello World" in Perl, but you can see it in Perl and many other languages at **http://www.freenetpages.co.uk/hp/alan.gauld/complang.htm**

running your first Perl script

- Perl is interpreted
  - you don't need to compile your scripts
- a variety of Perl binaries exist on our system
  - /usr/local/bin/perl (5.005)
  - /usr/local/bin/perl56 (5.6.1)
  - /usr/local/bin/perl58 (5.8.3)
  - /home/martink/perl/current/bin/perl (5.8.7)
- to check version "perl –V | head -1"

```
Summary of my perl5 (revision 5.0 version 8 subversion 3) configuration
```

```
> cat script.pl
print "Camels spit up to 10 meters, except in the US where they spit up to 33 feet.\n";

> /usr/bin/perl script.pl
Camels spit up to 10 meters, except in the US where they spit up to 33 feet.
```

#! notation

- you can specify the Perl binary within the script
  - this is the preferred way of doing it

```
#!/usr/bin/perl

print “I saw a smoking camel.\n”;
```

- you can pass flags to the Perl binary, if needed
  - we’ll cover useful flags later

```
#!/usr/bin/perl -w

print “I saw a smoking camel.\n”;
```

http://sunsite.uakom.sk/sunworldonline/swol-09-1999/swol-09-unix101.html

setting executable flag

- your shell will automatically execute “binaries” if their executable flag is set

```
> ls
-rw-r--r--    1 martink  users         112 2006-04-04 12:58 script.pl

> chmod +x script.pl
-rwxr-xr-x    1 martink  users         112 2006-04-04 12:58 script.pl

> which script.pl
./script.pl

> script.pl
Camels spit up to 10 meters, except in the US where they spit up to 33 feet.
```

- iterative script writing process
  - create/edit your script with your favourite text editor
  - set executable flag on with chmod (once)
  - run/debug script

choice of perl binaries

- on any large network, you will find many versions of the Perl interpreter (perl)
  - /usr/bin/perl – installed with the OS on the network node
  - /usr/local/bin/perl – installed for system-wide use, long long ago
  - /usr/local/bin/perlxxx – variety of links to other perl versions
- if you would like to play around
  - /usr/bin/perl
- if you are just starting and have no legacy dependancies
  - /usr/local/bin/perl58
  - additional modules may have been installed by systems
- if you need perl 5.6 for legacy use
  - /usr/local/bin/perl56

checking for version and binary compile settings

```
> /home/martink/bin/perl -V
Summary of my perl5 (revision 5 version 8 subversion 7) configuration:
  Platform:
    osname=linux, osvers=2.4.20-64gb-smp, archname=i686-linux-ld
    uname='linux xhost02 2.4.20-64gb-smp #1 smp wed aug 6 18:30:02 utc 2003 i686 unknown unknown gnulinux '
    config_args=''
    <...>
  Compiler:
    <...>
    intsize=4, longsize=4, ptrsize=4, doublesize=8, byteorder=1234
    d_longlong=define, longlongsize=8, d_longdbl=define, longdblsize=12
    ivtype='long', ivsize=4, nvtype='long double', nvsize=12, Off_t='off_t', lseeksize=8
    alignbytes=4, prototype=define
  Linker and Libraries:
   <...>
  Dynamic Linking:
   <...>

Characteristics of this binary (from libperl):
  Compile-time options: USE_LONG_DOUBLE USE_LARGE_FILES
  Built under linux
  Compiled at Sep 20 2005 16:19:46
  @INC:
    /home/martink/perl/5.8.7/lib/5.8.7/i686-linux-ld
    /home/martink/perl/5.8.7/lib/5.8.7
    /home/martink/perl/5.8.7/lib/site_perl/5.8.7/i686-linux-ld
    /home/martink/perl/5.8.7/lib/site_perl/5.8.7
    /home/martink/perl/5.8.7/lib/site_perl
    .
```

Perl variables

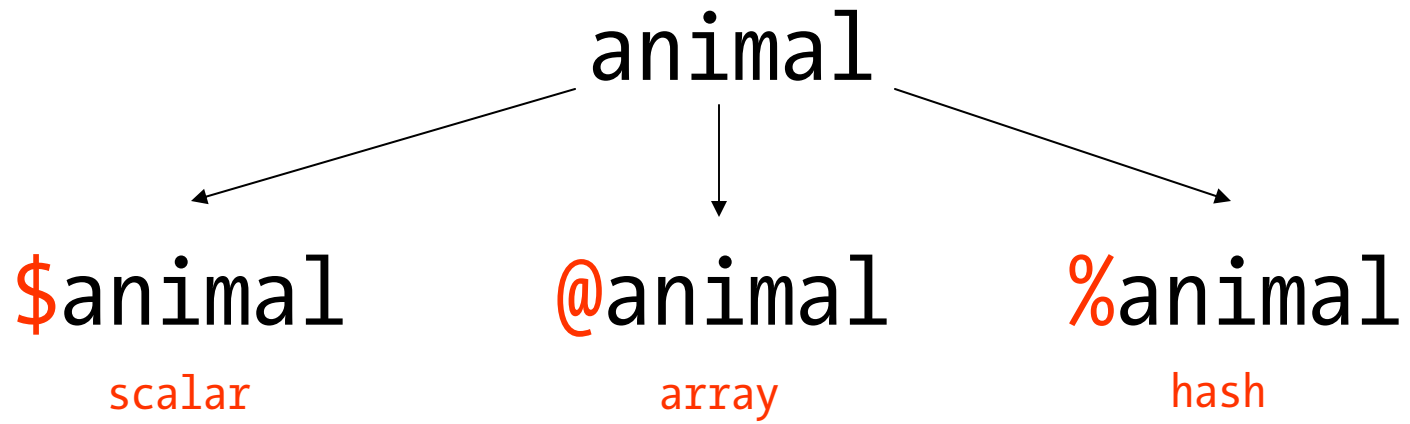
- Perl does not require that you specify what you want to store in a variable
  - this is a contrast to typed languages like C or Java
  - this is a boon and a bane – but you are in control, not the language

- the same variable, at different times, can hold
  - a number
  - a string
  - a letter
  - binary data

- Perl differentiates variables on the basis of plurality
  - a scalar variable holds a single value (a number, a string, a letter)
  - an array variable holds multiple values (a list of numbers, a list of strings)
    - a hash is a special type of array variable in which elements are indexed by strings, not integers

Perl variables

- Perl variables are preceded by a character that identifies the plurality of the variable



- you cannot access the value in a variable without using the appropriate prefix
- $animal, @animal and %animal are **different variables**
  - they are completely independent
  - they can hold different values

Perl variables

- you can name your variable whatever you want (mostly)
  - no special characters like $@% in variable names (obviously)
  - cannot begin with a number
  - no spaces

good	bad
$animal	
$animal123	
	$123animal
$big_animal	
	$big animal
$BigAnimal	
	$big;animal

- Perl is case sensitive
  - $animal $Animal and $ANIMAL are all different variables

Variable Assignment

- to give a variable a value, use =

```
$x = 1;
$y = 2;
$z = 3;

$x = $y = 0; # set both variables to zero

($x,$y,$z) = (1,2,3); # we'll see this later
```

- use undef to force a variable to become undefined

```
$x = 1;
$x = undef; # explicitly undefines $y
```

Scalars

- scalars are identified by $ and can hold only one value at a time
  - scalar is like a cup – if you want coffee, you need to remove the tea
  - arrays are more like icecube trays – you can have many icecubes

```
#!/usr/bin/perl

# a comment
$animal = “Camel”;
print $animal,”\n”;
$animal = “12 Camels”;
print $animal,”\n”;
$animal = 12; # an inline comment
print $animal,”\n”;


Camel
12 Camels
12
```

1. always specify where your perl interpreter is
2. each line must be terminated by a semicolon
3. \n codes for new line
4. # indicates the rest of the line is a comment

Basic Operators

- Perl has a lot of different operators – actions that you can apply to variables
  - unary/binary/trinary – operate on one/two/three scalars at a time
- Perl will try to do the right thing when you are operating on scalars
  - try mixing numbers and strings in an operation to see what happens

```
$w = "camel";
$x = 2;
$y = 3;

$z = $x + $y;  # 5
$z = $x * $y;  # 6
$z = $x / $y;  # 0.66666

$z = $w + $x;  # 2
$z = $w * $x;  # 0

$z = $w . $x;  # camel2
$z = $x . $y;  # 23
$z = $w + $w;  # 0
```

```
+ - * /     basic arithmetic
            2+3 = 5

**          exponentiation
            2**3 = 8

.           concatenation (period)
            2.3 = 23
```

Functions

- functions are things that Perl knows how to do out of the box
  - sqrt( )
  - sin( )
  - rand( ) – I will use rand() for a lot of examples
- you can write your own functions, of course

```
$x = rand(); # x is a random number uniformly sampled from [0,1)

print $x,"\n";
print sqrt($x),"\n";
print $x**$x,"\n";

0.0730786472558975
0.27033062581938
0.825975844619357
```

Flow Control

- Perl has a wide variety of branching operators
  - looping
  - condition checking
- let's learn the if conditional so we can write simple scripts

```
$x = rand();

if ( $x <= 0.5 ) {
  print $x . " is small\n";
} else {
  print $x . " is large\n";
}
```

```
if ( CONDITION ) {
  CODE
} else {
  CODE
}
```

- conditional operators
  - == tests for equality between numbers
  - eq tests for equality between strings

Many Perlisms Ahead

- Perl is about doing the same thing in a variety of ways
  - be creative
  - be stylish
  - be careful!

- start slowly and increase flair as necessary
  - make sure that, above else, you can understand your code!

```
if ( $x <= 0.5 ) {
  print $x,"\n";
}

if ( $x <= 0.5 ) { print $x,"\n"; }          if ( CONDITION ) { CODE }

print $x,"\n" if $x <= 0.5;                  CODE if CONDITION;
```

Interpolation

- interpolation can be a great source of frustration
- Perl tries to make it as painless as possible
- how a language interpolates variables is how a language decides how to evaluate strings, which may contain variables

```
$x = “Camel”;
$y = “I have a pet $x”;
print $y;

I have a pet Camel
```

- rule #1 – Perl interpolates variable values in double quotes

Interpolation – double quotes

- you can safely tuck your variables inside double quote and their values will be evaluated and inserted into the string

```
$x = “Camel”;

$y1 = “I have a pet $x”;
$y2 = “I have a pet “ . “$x”;
$y3 = “I have a pet “ . $x;
```

- variables will be interpolated, but no operations will be performed

```
$x = “Camel”;
$y = 2;
$z = 3;

$w = “I have a pet $x who told me $y times $z is $y*$z”;

I have a pet Camel who told me 2 times 3 is 2*3
```

Interpolation – double quotes

- if you want results of operations included in strings
  - concatenate them in
  - use temporary variables

```
$x = “Camel”;
$y = 2;
$z = 3;
$t = $y * $z;

$w1 = “I have a pet $x who told me $y times $z is $t”;
$w2 = “I have a pet $x who told me $y times $z is “ . $t;
$w2 = “I have a pet $x who told me $y times $z is “ . $y * $z;

I have a pet Camel who told me 2 times 3 is 6
```

Interpolation – single quotes

- no interpolation happens if you use single quotes

```
$x = "Camel";
$y = 2;
$z = 3;
$t = $y * $z;

$w1 = 'I have a pet $x who told me $y times $z is $t';

I have a pet $x who told me $y times $z is $t
```

- '$x' is a string that contains the characters "$" and "x", not the variable $x
  - you may want to print the text "$x" and not the value of the scalar x

```
$s1 = '$x';
$s3 = '$' . 'x';
$s2 = "\$" . "x"; # since $ is a special character, it needs to be escaped in double quotes
```

Interpolation – an *ation of pain

- you'll get used to Perl's own interpolation mechanism, but at first it can be frustrating

```
$x = "If I join the espresso club, I will save $2 on every coffee!";

If I join the espresso club, I will save  on every coffee!
```

what's going on? where's your money?

- you have just discovered the mysteries of Perl's special variables
  - special, as in hidden and confusing and impossible to remember
  - don't worry, we'll get to these shortly

- for now, if you have words or numbers preceeded by $ or @ or % in your strings, expect the unexpected!
  - don't worry, we'll sort these things out eventually

Interpolation

- understanding and getting a handle on interpolation is important because you'll be wanting to print things out
- Perl offers assistance in interpolating your strings
- think of the quotes " " as an operator, not as a container for a string
  - " " operates to replace all mention of variables with their values
  - ' ' operates to ignore all mention of variables and treats the string as a literal
- instead of quotes, you can use quote and quote-like operators

```
$x = "camel";
print "$x";        # camel
print qq(camel);   # camel
print qq($x);      # camel
print qq("$x");    # "camel"
```

qq(STRING) is equivalent to "STRING"

Uses of qq( )

- qq( ) helps you deal with strings which have quotes in them
  - remember, the qq(and) are the parts of the operator.

```
$x = “My camel’s name is “Bob””;

print $x

Bareword found where operator expected at ./script.pl line 21, near ""My camel's name is "Bob"
            (Missing operator before Bob?)
syntax error at ./script.pl line 21, near ""My camel's name is "Bob"
String found where operator expected at ./script.pl line 21, near "Bob"""
Execution of ./script.pl aborted due to compilation errors.
```

```
$x1 = “My camel’s name is \“Bob\””;
$x2 = qq{My camel’s name is “Bob”};
```

Flexibility of qq( )

- remember how I said Perl is flexible and gives you control
- how about flexible delimiters? now that's control!
  - non alpha-numeric, non whitespace

```
qq(My camel's name is "Bob");
qq{My camel's name is "Bob"};
qq/My camel's name is "Bob"/;
qq|My camel's name is "Bob"|;
qq$My camel's name is "Bob"$;
qq*My camel's name is "Bob"*;
qq!My camel's name is "Bob"!;
```

- you get the idea
  - there are other operators that have this flexibility
  - pick a delimiter and stick with it

```
qq/My camel's name is "Bob". His answer to $x is spitting and his favourite char is (/;
```

qq( ) vs q( )

- q( ) is equivalent to single quotes
  - all the flexibility of qq( ) without the interpolation

```
$x = 2;
q($x)     # $x
qq($x)    # 2
```

- if you have strings with lots of special characters, qq() and q() are a boon

${VAR}

- consider the following problem
  - you want to print out a variable and immediately another string after it

```
$x = 10;

print qq(Camels spit up to $xm);   # bad – no variable $xm

print qq(Camels spit up to ${x}m); # good

print “Camels spit up to ${x}m”;   # good

print “Camels spit up to $x”.”m”;  # good but messy

print “Camels spit up to $x \bm”;  # obfuscated
```

Interpolation Examples

- let's apply some of the things we've seen

```
$x = "x";
$X = "X";

print q($x) . qq( is the string "$x");
print q($X) . qq( is the string "$X");

$x is the string "x"
$X is the string "X"

if ( $x eq $X ) {
  print qq(\$x = $x and \$X = $X have the same contents);
} else {
  print qq(\$x = $x and \$X = $X are different);
}

$x = x and $X = X are different

print qq(If I get a new camel, I will name him $x.$X.$x);

If I get a new camel, I will name him x.X.x
```

Contratulations – you have conquered your fears

- you can now understand the following Perl line noise – see it's not that hard

```
$x = 1;

“$x.”                   1.
$x.”.”                  1.
qq($x).”.”              1.
qq(${x}.)               1.
qq($x.)                 1.

q!{${x}}!               {${x}}

“\$x$x.”                $x1.

“x${x}x”                x1x
“\$x${x}x”              $x1x

qq($x+${x}1+\$x)        1+11+$x
```

1.0.8.1.1

Introduction to Perl
Session 1

- you now know
  - all about scalars
  - all about interpolation
  - qq( ) and q( )
  - == and eq
  - **if** conditional
- next time
  - manipulating strings
  - regular expression basics