Inputs and Outputs

Aims

- Understand the main ways of obtaining data into a program that are not file/db based
- Have an idea about the two main outputs
- Be able to write a script which can take in an a user input, do something to it, and output it to screen

Inputs - ARGV

- The ARGV array is the array of arguments that you pass in on the command line

  bin/01-argv.pl

- Run the above script with some arguments

  >bin/01-argv.pl I love perl

  I

  love

  perl

Inputs - ARGV

- Our arguments can be obtained via array

  @ARGV

- Always assign this array to an array you define in your code!

  my @arguments = @ARGV;

Inputs - ARGV

- We can then do as we like with the array

```
foreach my $argument ( @ARGV ) {
  say $argument;
}
```

- This is by far the simplest, most convenient way of getting any data in. People expect it, you don't need to worry about interactivity.

Inputs - STDIN

- The command line is very nice. However, it's no good if you want to interact with your users
- For this we use the STDIN filehandle. A filehandle is a way that the program interacts with a source of data whilst running.
- STDIN is a filehandle for input from the keyboard (usually)

Inputs - STDIN

- Run

```
>bin/02-stdin.pl
Enter a number: 12
You entered 12. The square of this number is 144.
```

Inputs - STDIN

print 'Enter a number: ';

- We ask the user for something

  my $number = <STDIN>;

- <STDIN> is the filehandle, and tells perl to wait for something from the keyboard, followed by a newline (enter).
- We then must assign it to a variable, or it will be lost.

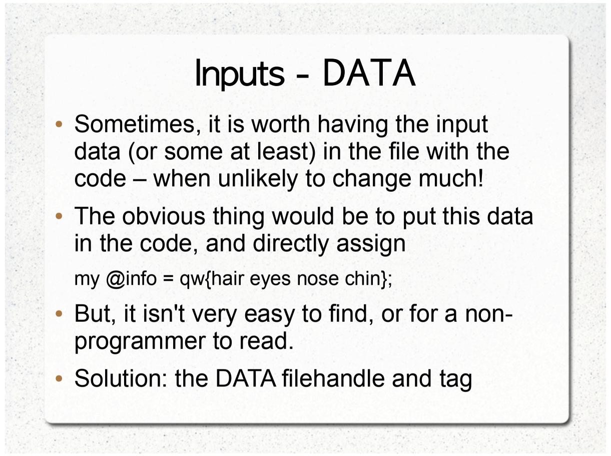
Inputs - STDIN

chomp $number;

- We take off the newline with the chomp method, since we want a number only

  *say "You entered $number. The square of this number is " . $number**2 . '.';*

- Feedback to the user their number, and do something with it.

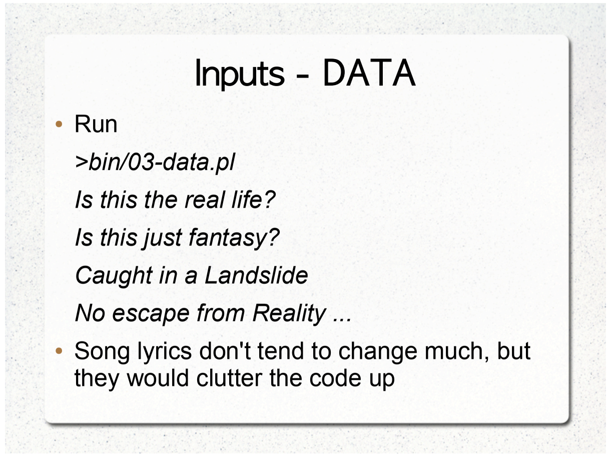


Inputs - DATA

- Sometimes, it is worth having the input data (or some at least) in the file with the code – when unlikely to change much!
- The obvious thing would be to put this data in the code, and directly assign

```
my @info = qw{hair eyes nose chin};
```

- But, it isn't very easy to find, or for a non-programmer to read.
- Solution: the DATA filehandle and tag

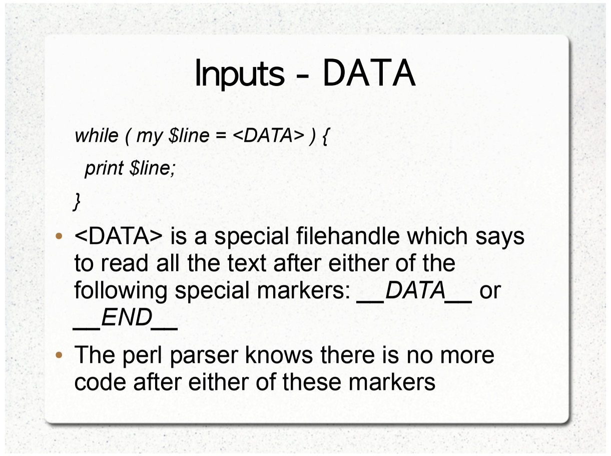


Inputs - DATA

- Run

```
>bin/03-data.pl
Is this the real life?
Is this just fantasy?
Caught in a Landslide
No escape from Reality ...
```

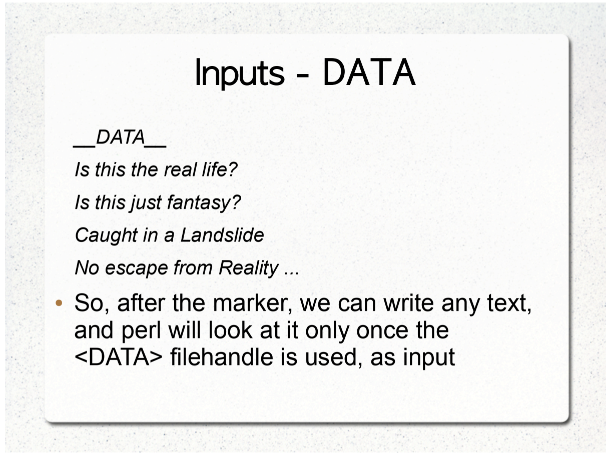
- Song lyrics don't tend to change much, but they would clutter the code up



Inputs - DATA

```
while ( my $line = <DATA> ) {
  print $line;
}
```

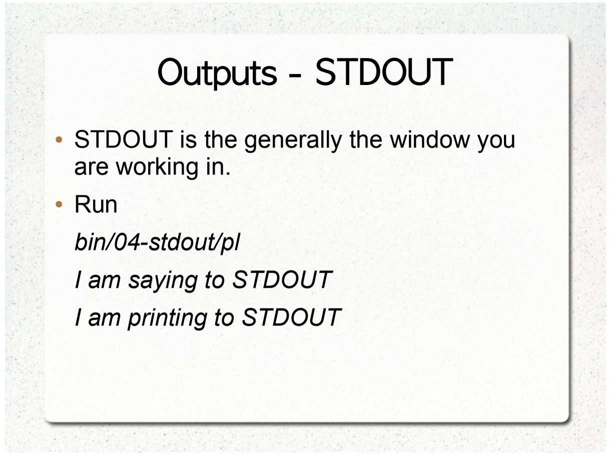
- <DATA> is a special filehandle which says to read all the text after either of the following special markers: *__DATA__* or *__END__*
- The perl parser knows there is no more code after either of these markers



Inputs - DATA

```
__DATA__
Is this the real life?
Is this just fantasy?
Caught in a Landslide
No escape from Reality ...
```

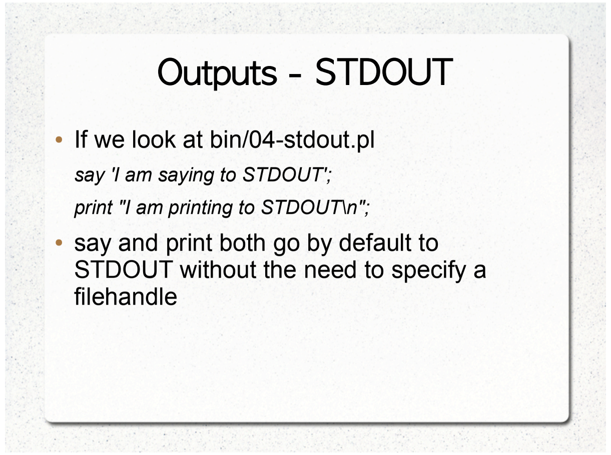
- So, after the marker, we can write any text, and perl will look at it only once the <DATA> filehandle is used, as input



Outputs - STDOUT

- STDOUT is the generally the window you are working in.
- Run

```
bin/04-stdout/pl
I am saying to STDOUT
I am printing to STDOUT
```

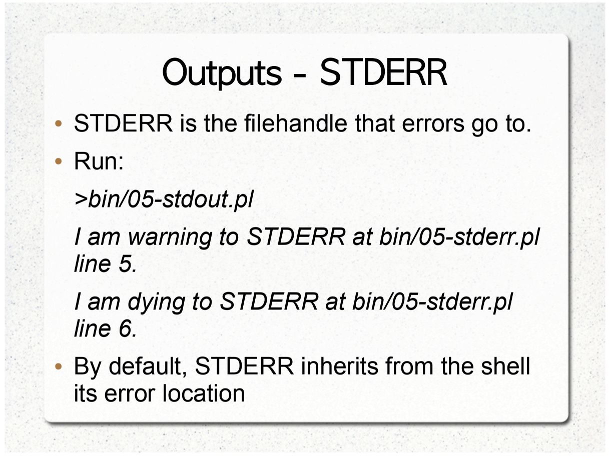


Outputs - STDOUT

- If we look at bin/04-stdout.pl

```
say 'I am saying to STDOUT';
print "I am printing to STDOUT\n";
```

- say and print both go by default to STDOUT without the need to specify a filehandle

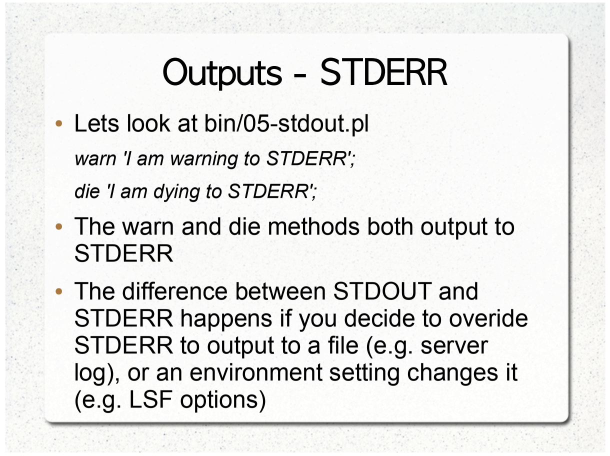


Outputs - STDERR

- STDERR is the filehandle that errors go to.
- Run:

```
>bin/05-stdout.pl
I am warning to STDERR at bin/05-stderr.pl line 5.
I am dying to STDERR at bin/05-stderr.pl line 6.
```

- By default, STDERR inherits from the shell its error location

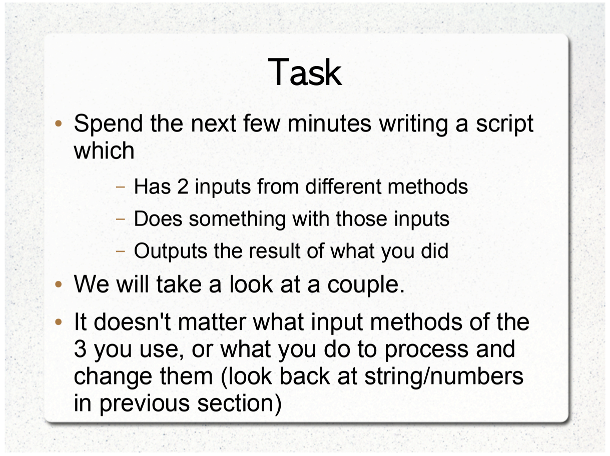


Outputs - STDERR

- Lets look at bin/05-stdout.pl

```
warn 'I am warning to STDERR';
die 'I am dying to STDERR';
```

- The warn and die methods both output to STDERR
- The difference between STDOUT and STDERR happens if you decide to overide STDERR to output to a file (e.g. server log), or an environment setting changes it (e.g. LSF options)



Task

- Spend the next few minutes writing a script which
  - Has 2 inputs from different methods
  - Does something with those inputs
  - Outputs the result of what you did
- We will take a look at a couple.
- It doesn't matter what input methods of the 3 you use, or what you do to process and change them (look back at string/numbers in previous section)