The Perl Debugger

- Perl has a built-in debugger.
- It runs perl scripts interactively, one command at a time.
- It will only run if your script is free of syntax errors.
- The debugger can help you find run-time errors.
- Also can run perl as an interactive shell
- These tools are excellent for exploring code snippets, regular expressions, finding bugs in code logic (=debugging)

```
Execute perl script normally with

perl myScript.pl

or

./myScript.pl

etc

Perl will try to run the script start to end.

Easiest way to run a script in the debugger is with

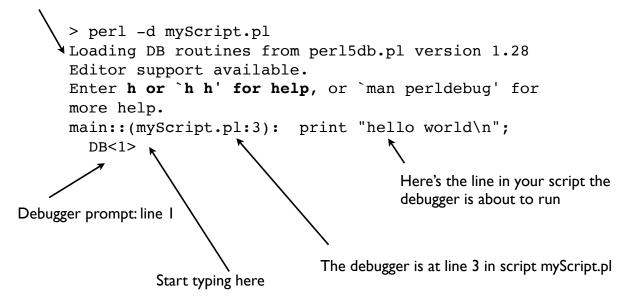
perl -d myScript.pl

You can add any command line parameters to the script as normal

perl -d myScript.pl infile.txt outfile.txt
```

The perl debugger

Startup lines, including info on getting help



Debugger: controlling program flow

```
h
            help
            quit
q
             next line or step through next line. Step will step down
n or s
into a subroutine and go through each line in the subroutine; next
goes to the next line.
<return>
               repeat last n or s
c 45
             continue to line 45
             break at line 45
b 45
b 45 $a == 0
                break at line 45 if $a equals 0
р $а
             print the value of $a
             unpack or extract the data structure in $a
x $a
R
            restart - useful if you think you have edited the script
             (pipe) print output one page at a time with more/less
|x @bigarray output big array in 'more/less' pager
```

x or unpack

- x expr evaluates expr in list context, returns formatted result.
- List context means elements are listed starting at index 0
- numbers are as is; strings are quoted and include new lines, tab characters etc.

```
DB<5> $a = 4
 DB<6> x $a
  4
 DB<10> @a = (5,10,15)
 DB<11> x @a
   5
0
1
   10
2
   15
  DB<16> a = foo\nboo
  DB<17> x $a
0 'foo
               note string goes across new line
boo'
```

Executing perl inside a script

You can type any perl line into the debugger

Script myScriptErrors.pl with logic errors and mistakes.

```
#!/usr/bin/perl
use warnings;
use strict;

my $data_line = 'ABC1\thuman\tMGTYIPLWQST\n';
my $sequence_index = 1;
my @fields = split /\t/, $data_line;
my $sequence = $fields[$sequence_index];
print "Protein sequence is: $sequence\n";

[Epinephrine:~] simonp% perl myScriptErrors.pl
Use of uninitialized value $sequence in concatenation
(.) or string at myScriptErrors.pl line 9.
Protein sequence is:
```

Debugger session to investigate run-time errors (1 of 2)

```
[Epinephrine:~] simonp% perl -d myScriptErrors.pl
Loading DB routines from per15db.pl version 1.33
Editor support available.
Enter h or `h h' for help, or `man perldebug' for more help.
main::(myScriptErrors.pl:5): my $data line = 'ABC1\thuman
\tMGTYIPLWQST\n';
 DB<1> n
main::(myScriptErrors.pl:6): my $sequence_index = 1;
 DB<1> p $data line
ABC1\thuman\tMGTYIPLWQST\n Problem!! We see \t not <tab> character
 DB<2>
main::(myScriptErrors.pl:7): my @fields = split /\t/, $data line;
 DB<2>
main::(myScriptErrors.pl:8): my $sequence = $fields[$sequence index];
 DB<2> x @fields
DB<3> n
main::(myScriptErrors.pl:9): print "Protein sequence is: $sequence
\n";
```

Debugger session (2 of 2)

```
DB<3> p $fields[$sequence_index]

This variable is empty

DB<4> p $sequence

So is this one

DB<5> n

Use of uninitialized value $sequence in concatenation (.) or string at myScriptErrors.pl line 9. Perl prints this warning at myScriptErrors.pl line 9

Protein sequence is:

Debugged program terminated. Use q to quit or R to restart, use o inhibit_exit to avoid stopping after program termination, h q, h R or h o to get additional info.

DB<5>
```

There are three problems with this script: i) double quotes needed in line 5; ii) sequence index should be 2 not 1; chomp needed after line 5

The interactive perl debugger

```
> perl -de 4
Loading DB routines from per15db.pl version 1.28
Editor support available.
Enter h or `h h' for help, or `man perldebug' for more help.
main::(-e:1):
  DB<1> $a = {foo => [1,2], boo => [2,3], moo => [6,7]}
  DB<2> x $a
0 HASH(0x8cd314)
   boo' => ARRAY(0x8c3298)
         2
      0
      1
   foo' \Rightarrow ARRAY(0x8d10d4)
      0
         1
   'moo' => ARRAY(0x815a88)
      0
      1
         7
```

Last words on the debugger

```
Inside the debugger,
use strict;
use warnings;
Behave a little differently from the way you are used to.
   -- if you print an undefined variable with 'p' or 'x' it will just be blank,
   you won't get a warning that you tried to print an uninitialized
   variable.
   -- if the script you are running in the debugger prints an uninitialized
   variable, you will get a warning.
Here's an example
    DB<1> p $a
                     No warning
    DB<2>
But you get a warning if the line you run comes from the script instead of
something you typed into the debugger
main::(someScriptErrors.pl:12): my $a;
  DB<1> n
main::(someScriptErrors.pl:13): print $a;
Use of uninitialized value $a in print at
someScriptErrors.pl line 13.
 at someScriptErrors.pl line 13
```