Thames Valley Perl Mongers

The Perl Debugger

David Pottage

Overview

- Perl Debugger
- Invoked with: perl -d <script>
- An interactive command line debugger
- No GUI
- Works remotely on any console.
- Built into perl. Nothing extra to install.
- Internal help & list of commands with '?'

Hello World

```
sub main
{
    my $opts=getArgs();

    if( exists $opts->{'user'} )
    {
        printf "Hello %s!\n", $opts->{'user'};
    }
    else
    {
        print "Hello World!\n";
    }

    return 0;
}

exit main();
```

Showing source code

- l Show next 10 lines of code
  - Breakable lines marked with a colon.
  - Or specify line numbers
- v Show lines around the current one
  - Or specify a line number.
- Both commands can be repeated to show more lines.

Step in, over & out.

- n Next code line. (step over)
  - Does not step into function calls.
- s Step execution.
  - Will step into function calls
- r Return from current function (step out)
  - Reports what the function returned & it's package.
  - This can be too verbose.

Debugger Control

- R Attempt to restart the debugger
  - Useful as breakpoints etc are retained.
  - Not always reliable.
  - Fails if the current directory has been changed.
  - Also fails if the command line has been changed.
- Command line history not working?
  - Install Term::ReadLine::Perl from CPAN or your linux distribution

Breakpoints and Variables

```
sub get_machines
{
    # parse /etc/ethers to get the list
    my %retHash;

    while (my $line = <DATA>)
    {
        my($mac, $ip, $name) = split /\s+/, $line;

        if( valid_ip($ip) )
        {
            $retHash{$name} = { 'mac'=>$mac, 'ip'=>$ip};
        }
    }

    return \%retHash;
}
```

Breakpoints

- b <line> Set a breakpoint on a line
  - Can also specify a condition
  - This is very flexible
- c <line> Continue to a line
  - Will stop earlier if a breakpoint is reached first.
- B <line> Remove a breakpoint
- L List all breakpoints

Examine variables with x

- x is the most versatile debugger command.
  - Dumps the contents of simple variables
  - Can be used to test bits of code like a shell
    - Regular expressions, functions calls, anything.
  - Hashes are output as arrays, so dereference them.
    - Eg: x \%hash

Complex Classes

```
use HTML::Element;
use HTML::TreeBuilder;

sub parseResPage
{
    My ( $rawHTML ) = @_;

    my $tree = HTML::TreeBuilder->new_from_content( $rawHTML );

    my @headings = $tree->look_down('_tag' => 'tr',
                                    'class' => 'post-head post_head');
    my $firstHeading = shift @headings;
    my $headText = $firstHeading->as_text();

    return $headText;
}
```

Limit recursive depth with x

- As before we can use x to show data
  - But it is not helpful because the screen fills with junk.
  - So use x [<depth>] <variable>
    - You will learn what depth is most helpful. I find 4 best for HTML::Element, or 3 for DBIC.
  - You can try out function calls to find what is correct.

Explicit breakpoints

```
package Hello::Controller::Root;
use Moose;
use namespace::autoclean;

BEGIN { extends 'Catalyst::Controller' }

sub hello :Global {
    my ( $self, $c, $user ) = @_;

    if( defined $user )
    {
        $c->response->body("Hello, ".$user."!");
    }
    else
    {
        $c->response->body("Hello, World!");
    }
}
```

Other ways to breakpoint

- Write c <fully qualified function name>
  - But the package must be loaded
  - I often put the FQN as a comment at the start of each function.
- Use DB::single=1 to add a breakpoint.
  - Works in any depth of code
  - Does not require the package to be loaded
  - But cannot be removed at runtime so can be annoying.

Other tips

- perl -demo
  - Gives you a shell to quickly evaluate stuff
- Windows users: run start perl -d script.
  - Prevents perl and cmd.com from fighting.
- Multi process programming
  - Use an xterm window on unix/linux, perl will create new windows for you.
  - But every thread hitting a breakpoint is a new window.
    - Including falling off the end.
    - Use DB::inhibit_exit =0

Alternatives

- Devel::ptkdb
- GUI debugger using Tk libraries
- Need to install the package from CPAN, or a distro package.
- Requires an X11 display.
  - Can be workable for remote debugging over ssh x forwarding.
- Hard to test expressions or regular expressions
- Shows you the stack, lets you examine variables further up.

Alternatives (2)

- Editor integrated debuggers
  - Vi
  - Eclipse EPIC
  - Many others
- Quality is variable
  - Sometimes hard to test out expressions
  - Some are unstable and will leave orphaned perl processes behind.
  - I tend not to use them, but don't let me stop you!

Questions