Devel::Confess

Generate stack traces for all errors and warnings

Graham Knop <haarg@haarg.org>
Perl “Exceptions”
Perl “Exceptions”

```perl
$ cat -n simple-error.pl
  1 package MyPackage {
  2
  3    sub my_function {
  4        die "there was an error";
  5    }
  6
  7    sub outer_function {
  8        my_function();
  9    }
 10 }
11
12 MyPackage::outer_function();

$ perl simple-error.pl
there was an error at simple-error.pl line 4.
```
Perl “Exceptions”

$ cat -n simple-error.pl
  1 package MyPackage {
  2
  3  sub my_function {
  4      die "there was an error";
  5  }
  6
  7  sub outer_function {
  8      my_function();
  9  }
 10 }
 11
 12 MyPackage::outer_function();
$ perl simple-error.pl
there was an error at simple-error.pl line 4.
Perl “Exceptions”

```perl
$ cat -n simple-error.pl
  1 package MyPackage {
  2
  3   sub my_function {
  4     die "there was an error";
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage::outer_function();
$ perl simple-error.pl
there was an error at simple-error.pl line 4.
```

Based on die location, not cause of error
Perl “Exceptions”

```perl
$ cat -n simple-error.pl
  1 package MyPackage {
  2
  3     sub my_function {
  4         die "there was an error";
  5     }
  6
  7     sub outer_function {
  8         my_function();
  9     }
 10 }
11
12 MyPackage::outer_function();

$ perl simple-error.pl
there was an error at simple-error.pl line 4.
```

Based on die location, not cause of error

Only one line of context
Perl “Exceptions”

$ cat -n simple-error.pl
  1 package MyPackage {
  2
  3    sub my_function {
  4        die "there was an error";
  5    }
  6
  7    sub outer_function {
  8        my_function();
  9    }
 10 }
11
12 MyPackage::outer_function();
$ perl simple-error.pl
there was an error at simple-error.pl line 4.
Carp::croak

$ cat -n croak-error.pl
   1 package MyPackage {
   2     use Carp;
   3     sub my_function {
   4         croak "there was an error";
   5     }
   6  }
   7  sub outer_function {
   8      my_function();
   9  }
 10 }
 11
 12 MyPackage::outer_function();
$ perl croak-error.pl
there was an error at croak-error.pl line 12.
Carp:::croak

$ cat -n croak-error.pl
  1  package MyPackage {
  2    use Carp;
  3    sub my_function {
  4      croak "there was an error";
  5    }
  6
  7    sub outer_function {
  8      my_function();
  9    }
 10  }
 11
  12  MyPackage:::outer_function();
$ perl croak-error.pl
there was an error at croak-error.pl line 12.
Carp:::croak

$ cat -n croak-error.pl
  1 package MyPackage {
  2   use Carp;
  3   sub my_function {
  4     croak "there was an error";
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage:::outer_function();

$ perl croak-error.pl
there was an error at croak-error.pl line 12.
$ cat -n croak-error.pl
  1 package MyPackage {
  2    use Carp;
  3    sub my_function {
  4       croak "there was an error";
  5    }
  6
  7    sub outer_function {
  8       my_function();
  9    }
 10 }
 11
 12 MyPackage::outer_function();
$ perl croak-error.pl
there was an error at croak-error.pl line 12.
```perl
$ cat -n confess-error.pl
  1 package MyPackage {
  2   use Carp;
  3   sub my_function {
  4     confess "there was an error";
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage::outer_function();

$ perl confess-error.pl
there was an error at confess-error.pl line 4.
   MyPackage::my_function() called at confess-error.pl line 8
   MyPackage::outer_function() called at confess-error.pl line 12
```
Carp::confess

```
$ cat -n confess-error.pl
  1 package MyPackage {
  2   use Carp;
  3   sub my_function {
  4     confess "there was an error";
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage::outer_function();
$ perl confess-error.pl
there was an error at confess-error.pl line 4.
   MyPackage::my_function() called at confess-error.pl line 8
   MyPackage::outer_function() called at confess-error.pl line 12
```
Carp::confess

$ cat -n confess-error.pl
  1 package MyPackage {
  2   use Carp;
  3   sub my_function {
  4     confess "there was an error";
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage::outer_function();

$ perl confess-error.pl
there was an error at confess-error.pl line 4.
  MyPackage::my_function() called at confess-error.pl line 8
  MyPackage::outer_function() called at confess-error.pl line 12
Carp::confess

$ cat -n confess-error.pl
 1 package MyPackage {
 2   use Carp;
 3   sub my_function {
 4     confess "there was an error";
 5   }
 6
 7   sub outer_function {
 8     my_function();
 9   }
10 }
11
12 MyPackage::outer_function();
$ perl confess-error.pl
there was an error at confess-error.pl line 4.
   MyPackage::my_function() called at confess-error.pl line 8
   MyPackage::outer_function() called at confess-error.pl line 12
But what if you don’t control the code?
$ cat -n simple-error.pl
  1 package MyPackage {
  2
  3  sub my_function {
  4    die "there was an error";
  5  }
  6
  7  sub outer_function {
  8    my_function();
  9  }
 10 }
11
 12 MyPackage::outer_function();
$ perl simple-error.pl
there was an error at simple-error.pl line 4.
$ cat -n simple-error.pl
1 package MyPackage {
2
3     sub my_function {
4         die "there was an error";
5     }
6
7     sub outer_function {
8         my_function();
9     }
10 }
11
12 MyPackage::outer_function();

$ perl simple-error.pl
there was an error at simple-error.pl line 4.
Carp::Always

$ cat -n simple-error.pl
1  package MyPackage {
2
3    sub my_function {
4      die "there was an error";
5    }
6
7    sub outer_function {
8      my_function();
9    }
10  }
11
12  MyPackage::outer_function();

$ perl -MCarp::Always simple-error.pl
there was an error at simple-error.pl line 4.
  MyPackage::my_function() called at simple-error.pl line 8
  MyPackage::outer_function() called at simple-error.pl line 12
Carp::Always

- Full stack traces for all types of errors
  - die, croak, confess
  - perl errors
  - Syntax errors
- Also for warnings
  - Including Carp::carp and Carp::cluck
Backtracking...
Perl Error Handling

• Most perl functions signal errors by return value
• Requires checking the return value
• A solution: autodie
$ cat -n autodie-error.pl
  1 package MyPackage {
  2     use autodie;
  3     sub my_function {
  4         open my $fh, '<', 'no-file';
  5     }
  6
  7     sub outer_function {
  8         my_function();
  9     }
 10 }
 11
 12 MyPackage::outer_function();
$ perl autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
at autodie-error.pl line 4
$ cat -n autodie-error.pl
  1 package MyPackage {
  2   use autodie;
  3   sub my_function {
  4     open my $fh, '<', 'no-file';
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
  12 MyPackage::outer_function();
$ perl autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
   at autodie-error.pl.pl line 4
```perl
package MyPackage {
    use autodie;

    sub my_function {
        open my $fh, '<', 'no-file';
    }

    sub outer_function {
        my_function();
    }

    MyPackage::outer_function();
}

$ Perl autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
    at autodie-error.pl line 4
```
package MyPackage {
    use autodie;
    sub my_function {
        open my $fh, '<', 'no-file';
    }

    sub outer_function {
        my_function();
    }
}

MyPackage::outer_function();

Can't open 'no-file' for reading: 'No such file or directory' at autodie-error.pl line 4
autodie

```
$ cat -n autodie-error.pl

```1 package MyPackage {
  2  use autodie;
  3  sub my_function {
  4      open my $fh, '<', 'no-file';
  5  }
  6
  7  sub outer_function {
  8      my_function();
  9  }
 10 }

 12 MyPackage::outer_function();

$ perl autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
at autodie-error.pl line 4
$ cat -n autodie-error.pl
  1 package MyPackage {
  2   use autodie;
  3   sub my_function {
  4     open my $fh, '<', 'no-file';
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage::outer_function();
$ perl autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
at autodie-error.pl line 4
$ cat -n autodie-error.pl
  1 package MyPackage {
  2   use autodie;
  3   sub my_function {
  4     open my $fh, '<', 'no-file';
  5   }
  6
  7   sub outer_function {
  8     my_function();
  9   }
 10 }
 11
 12 MyPackage::outer_function();

$ perl -MCarp::Always autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
at autodie-error.pl.pl line 4
$ cat -n autodie-error.pl
1 package MyPackage {
2   use autodie;
3   sub my_function {
4     open my $fh, '<', 'no-file';
5   }
6
7   sub outer_function {
8     my_function();
9   }
10 }
11
12 MyPackage::outer_function();
$ perl -MCarp::Always autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory'
at autodie-error.pl line 4
Exception Objects

- autodie and many others use objects rather than strings for errors
- Carp::Always appends stack trace to error string
- Can’t be done to an object without destroying it
- die, Carp, and Carp::Always pass objects through unmodified
Devel::Confess

- Based on Carp::Always code
- Also adds stack traces to exception objects
- And bare references
Devel::Confess with autodie

$ cat -n autodie-short.pl
  1 package MyPackage {
  2   use autodie;
  3   sub my_function { open my $fh, '<', 'no-file'; } 
  4   sub outer_function { my_function(); } 
  5 }
  6 MyPackage::outer_function();

$ perl -MCarp::Always autodie-short.pl
Can't open 'no-file' for reading: 'No such file or directory' at autodie-short.pl line 3
Devel::Confess with autodie

$ cat -n autodie-short.pl
    1 package MyPackage {
    2     use autodie;
    3     sub my_function { open my $fh, '<', 'no-file'; } 
    4     sub outer_function { my_function(); } 
    5 }
    6 MyPackage::outer_function();

$ perl -MDevel::Confess autodie-short.pl
Can't open 'no-file' for reading: 'No such file or directory' at autodie-short.pl line 3
    at (eval 11) line 154.
    MyPackage::__ANON__[(eval 7)[.../Fatal.pm:1676]:186]
    (GLOB(0x7fa22398d198), "<", "no-file") called at autodie-short.pl line 3
    MyPackage::my_function() called at autodie-short.pl line 4
    MyPackage::outer_function() called at autodie-short.pl line 6
$ cat -n autodie-short.pl
    1 package MyPackage {
    2   use autodie;
    3   sub my_function { open my $fh, '<', 'no-file'; }
    4   sub outer_function { my_function(); }
    5 }
    6 MyPackage::outer_function();
$ perl -MDevel::Confess autodie-short.pl
Can't open 'no-file' for reading: 'No such file or directory' at autodie-short.pl line 3
    at (eval 11) line 154.
    MyPackage::__ANON__[(eval 7)[.../Fatal.pm:1676]:186]
    (GLOB(0x7fa22398d198), "<", "no-file") called at autodie-short.pl line 3
    MyPackage::my_function() called at autodie-short.pl line 4
    MyPackage::outer_function() called at autodie-short.pl line 6
Devel::Confess via -d

```
$ cat -n autodie-short.pl
  1 package MyPackage {
  2    use autodie;
  3    sub my_function { open my $fh, '<', 'no-file'; }
  4    sub outer_function { my_function(); }
  5 }
  6 MyPackage::outer_function();
$ perl -d:Confess autodie-short.pl
Can't open 'no-file' for reading: 'No such file or directory' at autodie-short.pl line 3
  at (eval 11) line 154.
      MyPackage::__ANON__[(eval 7)[.../Fatal.pm:1676]:186]
      (GLOB(0x7fa22398d198), '<', "no-file") called at autodie-short.pl line 3
      MyPackage::my_function() called at autodie-short.pl line 4
      MyPackage::outer_function() called at autodie-short.pl line 6
```
Devel::Confess vs Carp::Always

- Includes stack traces for exception objects
- Extra context on anonymous subs and evals
- **Short** `-d:Confess` flag
- Can be disabled with `'no Devel::Confess;'`
- Fewer (test) prerequisites
Devel::Confess vs Carp::Always

• Includes stack traces for exception objects

• Extra context on anonymous subs and evals

• **Short** `-d:Confess` flag

• Can be disabled with `no Devel::Confess;`

• Fewer (test) prerequisites

• Extra features
Prior Art

Variations on Carp::Always

- Carp::Always
- Carp::Always::Color
- Carp::Always::Dump
- Carp::Source::Always
Extra Features: dump

Dump contents of references in arguments

$ perl -MCarp::Always call-with-ref.pl
Error occurred at call-with-ref.pl line 2.
   MyPackage::my_function(HASH(0x7ffde98032b8)) called at call-with-ref.pl line 3
   MyPackage::outer_function(HASH(0x7ffde98032b8)) called at call-with-ref.pl line 5

$ perl -d:Confess=dump call-with-ref.pl
Error occurred at call-with-ref.pl line 2.
   MyPackage::my_function({'hashref' => 'value'}) called at call-with-ref.pl line 3
   MyPackage::outer_function({'hashref' => 'value'}) called at call-with-ref.pl line 5
Extra Features: color

Highlight error and warning messages

$ perl -d:Confess=dump,color call-with-ref.pl
Error occurred at call-with-ref.pl line 2.
  MyPackage::my_function({'hashref' => 'value'}) called at call-with-ref.pl line 3
  MyPackage::outer_function({'hashref' => 'value'}) called at call-with-ref.pl line 5

$ perl -d:Confess=color warning.pl
Warning occurred at warning.pl line 2.
  MyPackage::my_function() called at warning.pl line 3
  MyPackage::outer_function() called at warning.pl line 5
Extra Features: builtin

Use built in traces for supported exception types

```
$ perl -d:Confess exception-class.pl
Error occurred at .../Exception/Class/Base.pm line 78.
  Exception::Class::Base::throw("MyException", "Error occurred")
called at exception-class.pl line 3
  MyPackage::my_function() called at exception-class.pl line 4
  MyPackage::outer_function() called at exception-class.pl line 6

$ perl -d:Confess=builtin exception-class.pl
Error occurred

Trace begun at exception-class.pl line 3
MyPackage::my_function at exception-class.pl line 4
MyPackage::outer_function at exception-class.pl line 6
```
$ perl -d:Confess=no-objects autodie-error.pl
Can't open 'no-file' for reading: 'No such file or directory' at (eval 7)[.../Fatal.pm:1676] line 154
$ perl -d:Confess=source short-error.pl
there was an error at short-error.pl line 2.
   MyPackage::my_function() called at short-error.pl line 3
   MyPackage::outer_function() called at short-error.pl line 5
===========================================================================
context for short-error.pl line 2:
  1: package MyPackage {
  2:   sub my_function { die "there was an error"; }
  3:   sub outer_function { my_function(); }
  4: }
  5: MyPackage::outer_function();
===========================================================================
context for short-error.pl line 3:
  1: package MyPackage {
  2:   sub my_function { die "there was an error"; }
  3:     sub outer_function { my_function(); }
  4: }
  5: MyPackage::outer_function();
===========================================================================
context for short-error.pl line 5:
  2:   sub my_function { die "there was an error"; }
  3:     sub outer_function { my_function(); }
  4: }
  5: MyPackage::outer_function();
===========================================================================
sub my_function { die "there was an error"; }
How does it work?
package Devel::Confess;
sub import {
    $SIG{__DIE__} = \&_die;
    $SIG{__WARN__} = \&_warn;
}
sub _warn {
    my @convert = _convert(@_);
    warn @convert;
}
sub _die {
    my @convert = _convert(@_);
    die @convert;
}
package Devel::Confess;
sub import {
    $SIG{__DIE__} = \&_die;
    $SIG{__WARN__} = \&_warn;
}
sub _warn {
    my @convert = _convert(@_);
    warn @convert;
}
sub _die {
    my @convert = _convert(@_);
    die @convert;
}
package Devel::Confess;
sub import {
    $SIG{__DIE__} = \&_die;
    $SIG{__WARN__} = \&_warn;
}
sub _warn {
    my @convert = _convert(@_);
    warn @convert;
}
sub _die {
    my @convert = _convert(@_);
    die @convert;
}
Internals: Overriding Errors

__WARN__ handlers silence warnings unless the handler outputs them.
package Devel::Confess;
sub import {
    $SIG{__DIE__} = \&_die;
    $SIG{__WARN__} = \&_warn;
}
sub _warn {
    my @convert = _convert(@_);
    warn @convert;
}
sub _die {
    my @convert = _convert(@_);
    die @convert;
}
Attaching to Objects

- Generate a subclass of the exception object
- Subclass will know how to attach stack trace
- Re-bless exception into subclass
my $pack_suffix = 'A000';
my %attached;
sub _convert {
    if (my $class = blessed(my $ex = $_[0]))) {
        my $id = refaddr($ex);
        my $message = _stack_trace;
        $attached{$id} = [ $ex, $class, $message ];
        weaken $attached{$id}[0];
        my $newclass = __PACKAGE__::__ANON__.$pack_suffix++::__;
        no strict 'refs';
        @{$newclass::_('ISA')} = ('Devel::Confess::_Attached', $class);
        bless $ex, $newclass;
    }
}
my $pack_suffix = 'A000';
my %attached;

sub _convert {
    if (my $class = blessed(my $ex = $_[0]))) {
        my $id = refaddr($ex);
        my $message = _stack_trace();
        $attached{$id} = [ $ex, $class, $message ];
        weaken $attached{$id}[0];
        my $newclass = __PACKAGE__::__ANON__.$pack_suffix++. '__';
        no strict 'refs';
        @{$newclass::__ISA'} = ('Devel::Confess::__Attached', $class);
        bless $ex, $newclass;
    }
}
my $pack_suffix = 'A000';
my %attached;
sub _convert {
  if (my $class = blessed(my $ex = $_[0])) {
    my $id = refaddr($ex);
    my $message = _stack_trace();
    $attached{$id} = [ $ex, $class, $message ];
    weaken $attached{$id}[0];
    my $newclass = __PACKAGE__::__ANON__.$pack_suffix++.__';
  }
  no strict 'refs';
  @{$newclass.'::ISA'} = ('Devel::Confess::_Attached', $class);
  bless $ex, $newclass;
}

Internals: Re-blessing

```perl
my $pack_suffix = 'A000';
my %attached;
sub _convert {
  if (my $class = blessed(my $ex = $_[0]))) {
    my $id = refaddr($ex);
    my $message = _stack_trace();
    $attached{$id} = [ $ex, $class, $message ];
    weaken $attached{$id}[0];
    my $newclass = __PACKAGE__::__ANON__.$pack_suffix++.__;
    no strict 'refs';
    @{$newclass::ISA} = ('Devel::Confess::_Attached', $class);
    bless $ex, $newclass;
  }
}
```

Generate new unique class name

Inheriting from the original class, and a class that will attach the stack trace
my $pack_suffix = 'A000';
my %attached;
sub _convert {
    if (my $class = blessed(my $ex = $_[0]))) {
        my $id = refaddr($ex);
        my $message = _stack_trace();
        $attached{$id} = [ $ex, $class, $message ];
        weaken $attached{$id};
        my $newclass = __PACKAGE__::__ANON__.$pack_suffix++. '__';
        no strict 'refs';
        @{$newclass '::ISA'} = ('Devel::Confess::_Attached', $class);
        bless $ex, $newclass;
    }
}
{  
  package Devel::Confess::_Attached;
  use overload '"' => sub {
    return join('', Devel::Confess::_ex_as_strings(@_));
  };
}

sub _ex_as_strings {
  my ($ex, $class, $trace) = @{$attached{refaddr $_[0]}};
  my $newclass = ref $ex;
  bless $ex, $class;
  my $out = "$ex";
  bless $ex, $newclass;
  return ($out, $trace);
}
Overload stringification, combining object’s original string form with stack trace
{  
    package Devel::Confess::_Attached;
    use overload '""""' => sub {  
        return join('', Devel::Confess::_ex_as_strings(@_));  
    };
}

sub _ex_as_strings {  
    my ($ex, $class, $trace) = @{$attached{refaddr $_[0]} };  
    my $newclass = ref $ex;
    bless $ex, $class;
    my $out = "$ex";
    bless $ex, $newclass;
    return ($out, $trace);
}
Internals: Stringifying

Trigger object’s original stringification by blessing into original class, then re-re-blessing to our generated class

```perl
package Devel::Confess::_Attached;
use overload '' => sub {
    return join('', Devel::Confess::_ex_as_strings(@_));
};

sub _ex_as_strings {
    my ($ex, $class, $trace) = @{$attached{refaddr $_[0]});
    my $newclass = ref $ex;
    bless $ex, $class;
    my $out = "$ex";
    bless $ex, $newclass;
    return ($out, $trace);
}
```
perl’s -d flag

- Runs the perl debugger: perl5db.pl
- Enables all debugging flags in $^P
- -d:<Module> specifies an alternate implementation
- Loads Devel::<Module>, then runs based on flags
- Line by line debugging: DB::DB() is called for every line of code run
Devel::Confess via -d

- Devel::Confess isn’t a debugger
- Has no use for most debugging flags
- Instead of creating a DB::DB() sub, disable the flags
- ... except turn on informative names for anonymous subs and evals
- Flags can be enabled even when not run with -d
if (!defined &DB::DB && $^P & 0x02) {
    $^P = 0;
}
sub import {
    $^P |= 0x100 | 0x200;
}
Internals: Debug Flags

if (!defined &DB::DB && $^P & 0x02) {
    $^P = 0;
}

sub import {
    $^P |= 0x100 | 0x200;
}

Line by line debugging enabled, but required sub not defined
Internals: Debug Flags

```perl
if (!defined &DB::DB && $^P & 0x02) {
  $^P = 0;
}
sub import {
  $^P |= 0x100 | 0x200;
}
```

Line by line debugging enabled, but required sub not defined

Loaded by -d, Disable all debugging features
Internals: Debug Flags

```perl
if (!defined &DB::DB && $^P & 0x02) {
    $^P = 0;
}

sub import {
    $^P |= 0x100 | 0x200;
}
```

Provide extra context for `evals` and `anonymous subs`
package Devel::Confess::Source;

$^P |= $^V >= 5.010 ? 0x400 : do {
  *DB::DB = sub {}
  unless defined &DB::DB;
  0x02;
};

sub _get_content {
  my $file = shift;
  no strict 'refs';
  if (exists $::{'_<'.}$file) && @{$::{'_<'.}$file}) {
    return @{$::{'_<'.}$file};
  }
}
package Devel::Confess::Source;

$^P |= $^V >= 5.010 ? 0x400 : do {
    *DB::DB = sub {}
    unless defined &DB::DB;
    0x02;
};

sub _get_content {
    my $file = shift;
    no strict 'refs';
    if (exists $::{'_<'.}$file) && @{ ':::_<'.}$file }) {
        return @{ ':::_<'.}$file ;
    }
}
package Devel::Confess::Source;

$^P |= $^V >= 5.010 ? 0x400 : do {
    *DB::DB = sub {};

    unless defined &DB::DB;

    0x02;
};

sub _get_content {
    my $file = shift;
    no strict 'refs';
    if (exists $::{'_<'.}$file && @{':_<'.}$file) {
        return \@{ '_<'.}$file;
    }
}

Flag for saving source

Line by line debugging will also save source where
0x400 is unsupported
package Devel::Confess::Source;

$^P |= $^V >= 5.010 ? 0x400 : do {
  *DB::DB = sub {}
    unless defined &DB::DB;
  0x02;
};

sub _get_content {
  my $file = shift;
  no strict 'refs';
  if (exists $::{'_<'. $file} && @{ '::_<'. $file }) {
    return @{ '::_<'. $file }
  }
}

Magic array where source is saved
Why a fork?

- Greater potential for errors
- Re-blessing objects can interfere with naive code
- Code complexity
  - Carp::Always is incredibly simple
  - Author prefers to keep it that way
  - 40 lines of code vs 470+
Devel::Confess

- [https://metacpan.org/pod/Devel::Confess](https://metacpan.org/pod/Devel::Confess)
- [https://github.com/haarg/Devel-Confess](https://github.com/haarg/Devel-Confess)
- Thanks to Adriano Ferreira for Carp::Always

Graham Knop <haarg@haarg.org>