Devel::Confess

Generate stack traces for all errors and warnings

Graham Knop <haarg@haarg.org>

Perl “Exceptions”

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”

```
$ 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.
```

Based on die location, not cause of error

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.
```

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.
```

Based on location that caused error

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::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
```

Full stack trace

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?

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 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 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

Carp::Always with DBIx::Class

```
$ cat -n dbic-error.pl
     1 package MyPackage {
     2   use DBIx::Class::Schema;
     3   sub my_function {
     4     DBIx::Class::Schema->connect->storage->ensure_connected;
     5   }
     6
     7   sub outer_function {
     8     my_function();
     9   }
    10 }
    11
    12 MyPackage::outer_function();
$ perl autodie-error.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-error.pl line 4
```

Carp::Always with DBIx::Class

```
$ cat -n dbic-error.pl
     1 package MyPackage {
     2   use DBIx::Class::Schema;
     3   sub my_function {
     4     DBIx::Class::Schema->connect->storage->ensure_connected;
     5   }
     6
     7   sub outer_function {
     8     my_function();
     9   }
    10 }
    11
    12 MyPackage::outer_function();
$ perl -MCarp::Always autodie-error.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-error.pl line 4
```

Carp::Always with DBIx::Class

```
$ cat -n dbic-error.pl
     1 package MyPackage {
     2   use DBIx::Class::Schema;
     3   sub my_function {
     4     DBIx::Class::Schema->connect->storage->ensure_connected;
     5   }
     6
     7   sub outer_function {
     8     my_function();
     9   }
    10 }
    11
    12 MyPackage::outer_function();
$ perl -MCarp::Always autodie-error.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-error.pl line 4
```

No extra context?

Exception Objects

- `DBIx::Class` 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` concept
- Also adds stack traces to exception objects
- And bare references

Devel::Confess with DBIC

```
$ cat -n dbic-short.pl
    1 package MyPackage {
    2   use DBIx::Class::Schema;
    3   sub my_function { DBIx::Class::Schema->connect->storage->…
    4   sub outer_function { my_function(); }
    5 }
    6 MyPackage::outer_function();
$ perl -MCarp::Always dbic-short.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-short.pl line 3
```

Devel::Confess with DBIC

```
$ cat -n dbic-short.pl
     1 package MyPackage {
     2   use DBIx::Class::Schema;
     3   sub my_function { DBIx::Class::Schema->connect->storage->…
     4   sub outer_function { my_function(); }
     5 }
     6 MyPackage::outer_function();
$ perl -MDevel::Confess dbic-short.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-short.pl line 3
 at DBIx/Class/Schema.pm line 1081.
  DBIx::Class::Schema::throw_exception(DBIx::Class::Schema=HASH(0x
7fff09803438), "You did not provide any connection_info") called at
DBIx/Class/Storage.pm line 113
  DBIx::Class::Storage::throw_exception(DBIx::Class::Storage::DBI=H
ASH(0x7fff09a8be88), "You did not provide any connection_info") cal
```

Devel::Confess via -d

```
$ cat -n dbic-short.pl
     1 package MyPackage {
     2   use DBIx::Class::Schema;
     3   sub my_function { DBIx::Class::Schema->connect->storage->…
     4   sub outer_function { my_function(); }
     5 }
     6 MyPackage::outer_function();
$ perl -MDevel::Confess dbic-short.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-short.pl line 3
 at DBIx/Class/Schema.pm line 1081.
  DBIx::Class::Schema::throw_exception(DBIx::Class::Schema=HASH(0x
7fff09803438), "You did not provide any connection_info") called at
DBIx/Class/Storage.pm line 113
  DBIx::Class::Storage::throw_exception(DBIx::Class::Storage::DBI=H
ASH(0x7fff09a8be88), "You did not provide any connection_info") cal
```

Devel::Confess via -d

```
$ cat -n dbic-short.pl
     1 package MyPackage {
     2   use DBIx::Class::Schema;
     3   sub my_function { DBIx::Class::Schema->connect->storage->…
     4   sub outer_function { my_function(); }
     5 }
     6 MyPackage::outer_function();
$ perl -d:Confess dbic-short.pl
DBIx::Class::Storage::DBI::_connect(): You did not provide any
connection_info at dbic-short.pl line 3
 at DBIx/Class/Schema.pm line 1081.
  DBIx::Class::Schema::throw_exception(DBIx::Class::Schema=HASH(0x
7fff09803438), "You did not provide any connection_info") called at
DBIx/Class/Storage.pm line 113
  DBIx::Class::Storage::throw_exception(DBIx::Class::Storage::DBI=H
ASH(0x7fff09a8be88), "You did not provide any connection_info") cal
```

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
```

Extra Features: objects

Features can be disabled using a ‘`no-`‘ prefix

```
$ 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
```

Extra Features: source

```
$ 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();
================================================================================
```

How does it work?

Internals: Overriding Errors

```
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

```
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;
}
```

__DIE__ handler is called for all errors (user or internal)

Internals: Overriding Errors

```
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;
}
```

__DIE__ handler is called for all errors (user or internal)

__WARN__ handler does the same for warnings

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;
}
```

Internals: Overriding Errors

__WARN__ handlers silence warnings unless the handler outputs them

__DIE__ handlers can't 'silence' errors, but they can override 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

Internals: Re-blessing

```
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

Save stack trace external to object

```
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

```
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

Internals: Re-blessing

```
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

Internals: Re-blessing

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

Bless into new generated class

Internals: Stringifying

```
{
  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

```
{
  package Devel::Confess::_Attached;
  use overload '""' => sub {
    return join('', Devel::Confess::_ex_as_strings(@_));
  };
}
sub _ex_as_st
  my ($ex, $c                                    $_[0]}};
  my $newclas
  bless $ex,
  my $out = "$ex";
  bless $ex, $newclass;
  return ($out, $trace);
}
```

Overload stringification, combining object's original string form with stack trace

Internals: Stringifying

```
{
  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);
}
```

Stack trace from external storage

Internals: Stringifying

```
{
  package Devel::Confess::_Attached;
  use ove
    retur
  };
}
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);
}
```

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

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://github.com/haarg/Devel-Confess
- Thanks to Adriano Ferreira for Carp::Always

Graham Knop <haarg@haarg.org>