

**NAME**

Sub::Quote – Efficient generation of subroutines via string eval

**SYNOPSIS**

```
package Silly;

use Sub::Quote qw(quote_sub unquote_sub quoted_from_sub);

quote_sub 'Silly::kitty', q{ print "meow" };

quote_sub 'Silly::doggy', q{ print "woof" };

my $sound = 0;

quote_sub 'Silly::dagron',
  q{ print ++$sound % 2 ? 'burninate' : 'roar' },
  { '$sound' => \ $sound };

And elsewhere:

Silly->kitty;   # meow
Silly->doggy;   # woof
Silly->dagron;  # burninate
Silly->dagron;  # roar
Silly->dagron;  # burninate
```

**DESCRIPTION**

This package provides performant ways to generate subroutines from strings.

**SUBROUTINES****quote\_sub**

```
my $coderef = quote_sub 'Foo::bar', q{ print $x++ . "\n" }, { '$x' => \0 };
```

Arguments: ?\$name, \$code, ?%captures, ?%options

\$name is the subroutine where the coderef will be installed.

\$code is a string that will be turned into code.

%captures is a hashref of variables that will be made available to the code. The keys should be the full name of the variable to be made available, including the sigil. The values should be references to the values. The variables will contain copies of the values. See the “SYNOPSIS”’s `Silly::dagron` for an example using captures.

Exported by default.

*options***no\_install**

**Boolean.** Set this option to not install the generated coderef into the passed subroutine name on undefer.

**no\_defer**

**Boolean.** Prevents a `Sub::Defer` wrapper from being generated for the quoted sub. If the sub will most likely be called at some point, setting this is a good idea. For a sub that will most likely be inlined, it is not recommended.

**package**

The package that the quoted sub will be evaluated in. If not specified, the package from sub calling `quote_sub` will be used.

**hints**

The value of `$^H` to use for the code being evaluated. This captures the settings of the strict pragma. If not specified, the value from the calling code will be used.

**warning\_bits**

The value of  `${^WARNING_BITS}`  to use for the code being evaluated. This captures the warnings set. If not specified, the warnings from the calling code will be used.

**%^H**

The value of  `%^H`  to use for the code being evaluated. This captures additional pragma settings. If not specified, the value from the calling code will be used if possible (on perl 5.10+).

**attributes**

The “Subroutine Attributes” in  `perlsub`  to apply to the sub generated. Should be specified as an array reference. The attributes will be applied to both the generated sub and the deferred wrapper, if one is used.

**file**

The apparent filename to use for the code being evaluated.

**line**

The apparent line number to use for the code being evaluated.

**unquote\_sub**

```
my $coderef = unquote_sub $sub;
```

Forcibly replace subroutine with actual code.

If  `$sub`  is not a quoted sub, this is a no-op.

Exported by default.

**quoted\_from\_sub**

```
my $data = quoted_from_sub $sub;
```

```
my ($name, $code, $captures, $compiled_sub) = @$data;
```

Returns original arguments to  `quote_sub` , plus the compiled version if this sub has already been unquoted.

Note that  `$sub`  can be either the original quoted version or the compiled version for convenience.

Exported by default.

**inlinify**

```
my $prelude = capture_unroll '$captures', {
    '$x' => 1,
    '$y' => 2,
}, 4;
```

```
my $inlined_code = inlinify q{
    my ($x, $y) = @_;
```

```
    print $x + $y . "\n";
}, '$x, $y', $prelude;
```

Takes a string of code, a string of arguments, a string of code which acts as a “prelude”, and a **Boolean** representing whether or not to localize the arguments.

**quotify**

```
my $quoted_value = quotify $value;
```

Quotes a single (non-reference) scalar value for use in a code string. The result should reproduce the original value, including strings, undef, integers, and floating point numbers. The resulting floating point numbers (including infinities and not a number) should be precisely equal to the original, if possible. The exact format of the resulting number should not be relied on, as it may include hex floats or math expressions.

**capture\_unroll**

```
my $prelude = capture_unroll '$captures', {
    '$x' => 1,
    '$y' => 2,
}, 4;
```

Arguments: `$from`, `\%captures`, `$indent`

Generates a snippet of code which is suitable to be used as a prelude for “`inlinify`”. `$from` is a string will be used as a hashref in the resulting code. The keys of `%captures` are the names of the variables and the values are ignored. `$indent` is the number of spaces to indent the result by.

**qsub**

```
my $hash = {
    coderef => qsub q{ print "hello"; },
    other   => 5,
};
```

Arguments: `$code`

Works exactly like “`quote_sub`”, but includes a prototype to only accept a single parameter. This makes it easier to include in hash structures or lists.

Exported by default.

**sanitize\_identifier**

```
my $var_name = '$variable_for_' . sanitize_identifier('@name');
quote_sub qq{ print \${$var_name} }, { $var_name => \ $value };
```

Arguments: `$identifier`

Sanitizes a value so that it can be used in an identifier.

**ENVIRONMENT****SUB\_QUOTE\_DEBUG**

Causes code to be output to `STDERR` before being eval'd. Several forms are supported:

1 All subs will be output.

`/foo/`

Subs will be output if their code matches the given regular expression.

`simple_identifier`

Any sub with the given name will be output.

`Full:::identifier`

A sub matching the full name will be output.

`Package::Name::`

Any sub in the given package (including anonymous subs) will be output.

**CAVEATS**

Much of this is just string-based code-generation, and as a result, a few caveats apply.

**return**

Calling `return` from a `quote_sub`'ed sub will not likely do what you intend. Instead of returning from the code you defined in `quote_sub`, it will return from the overall function it is composited into.

So when you pass in:

```
quote_sub q{ return 1 if $condition; $morecode }
```

It might turn up in the intended context as follows:

```
sub foo {

    <important code a>
```

```

do {
    return 1 if $condition;
    $morecode
};
<important code b>

}

```

Which will obviously return from foo, when all you meant to do was return from the code context in quote\_sub and proceed with running important code b.

### pragmas

Sub::Quote preserves the environment of the code creating the quoted subs. This includes the package, strict, warnings, and any other lexical pragmas. This is done by prefixing the code with a block that sets up a matching environment. When inlining Sub::Quote subs, care should be taken that user pragmas won't effect the rest of the code.

## SUPPORT

Users' IRC: #moose on irc.perl.org

Development and contribution IRC: #web-simple on irc.perl.org

Bugtracker: <<https://rt.cpan.org/Public/Dist/Display.html?Name=Sub-Quote>>

Git repository: <[git://github.com/moose/Sub-Quote](https://github.com/moose/Sub-Quote).git>

Git browser: <<https://github.com/moose/Sub-Quote>>

## AUTHOR

mst – Matt S. Trout (cpan:MSTROUT) <mst@shadowcat.co.uk>

## CONTRIBUTORS

frew – Arthur Axel “fREW” Schmidt (cpan:FREW) <frioux@gmail.com>

ribasushi – Peter Rabbitson (cpan:RIBASUSHI) <ribasushi@cpan.org>

Mithaldu – Christian Walde (cpan:MITHALDU) <walde.christian@googlemail.com>

tobyink – Toby Inkster (cpan:TOBYINK) <tobyink@cpan.org>

haarg – Graham Knop (cpan:HAARG) <haarg@cpan.org>

bluefeet – Aran Deltac (cpan:BLUEFEET) <bluefeet@gmail.com>

ether – Karen Etheridge (cpan:ETHER) <ether@cpan.org>

dolmen – Olivier Mengué (cpan:DOLMEN) <dolmen@cpan.org>

alexbio – Alessandro Ghedini (cpan:ALEXBIO) <alexbio@cpan.org>

getty – Torsten Raudssus (cpan:GETTY) <torsten@raudss.us>

arcanez – Justin Hunter (cpan:ARCANEZ) <justin.d.hunter@gmail.com>

kanashiro – Lucas Kanashiro (cpan:KANASHIRO) <kanashiro.duarte@gmail.com>

djerius – Diab Jerius (cpan:DJERIUS) <djerius@cfa.harvard.edu>

## COPYRIGHT

Copyright (c) 2010–2016 the Sub::Quote “AUTHOR” and “CONTRIBUTORS” as listed above.

## LICENSE

This library is free software and may be distributed under the same terms as perl itself. See <<http://dev.perl.org/licenses/>>.