On Topic

Exploring "topic" and "topicalizers" in Perl 6

Allison Randal
University of Portland
YAPC::NA 2002
"topic" and "topicalizer"

This is not Vogon poetry.

It's linguistics…

…which might be worse.

Be brave.
• Every discourse has a topic.
• Topic is "what we're talking about".
• The topic of this talk is "Topic".
• If you overheard:

"I saw Lister yesterday."
"Really? What's he up to these days?"
"Oh, you know, drunk again, and mooning over that awful Krissy Kochanski."
etc., ad nauseum...

• You would know the topic was "Lister".
A topicalizer flags the current topic.

Some topicalizers in English:

For our first trick tonight, ladies and gentlemen, my partner Kryten will attempt to eat a boiled egg.

Given that God is infinite, and that the universe is also infinite, would you like a toasted tea-cake?

Regarding topicalizers, I should point out that this sentence starts with one.
• Topic is the most important variable in a block of code.

• Really, the underlying data structure is the topic, not the variable.
Topic in Perl

- Variables are just names for storage locations.
- Multiple variables may be aliased to the same storage location.
  
  "Rimmer", "he", "the hologram", "Smeghead"
  
  
  \$_, $name, %characters{'title'}

- If the topic has more than one alias, all are the current topic.
Why learn about topic?

It's not required.

The first law of topic: "Topic is \$_".
To use topic, use `$_`:

```perl
top
chomp;
s///;
when condition { ... }
.method_call;
```

You don't have to understand topic, but you might want to.
Topicalizers in Perl

• A topicalizer flags the current topic.

• A quick summary of topicalizers:

  given  bare closures
  for    =~
  ->     etc...
  method
  rule
  CATCH
The simplest topicalizer is `given`.

```plaintext
given $name {
  when "Lister" {
    print "Smeg!";
  }
  when "Cat" {
    print "Orange?! With this suit?!";
  }
  when "Rimmer" {
    print "4,691 irradiated haggis.";
  }
}
```
Fruit Loops and M&M's

- The classic topicalizer is `for`.

```plaintext
for @orders {
    when /scone/ {
        print "Would you like some toast?"
    }
    when /croissant/ {
        print "Hot, buttered, scrummy toast?"
    }
    when /toast/ {
        print "Really? How about a muffin?"
    }
}
```
To rw or not to rw...

- In this simple form both `for` and `given` create `$_` as `rw`.

```perl
for @names {
    chomp;
    s:w/Arnold J\.//;
    s:w/Dave //;
}
```
• The most flexible topicalizer is \(-\rightarrow\).

• By itself, it defines an anonymous sub:

```perl
$cleanup = -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}

$intro = "Fear not, I’m Captain Rimmer!";
$cleanup($intro);
```
To rw or not to rw... (cont.)

- creates its aliases read-only.

- Unless `rw` is specified.

```perl
$cleanup = -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}
```
Bow and Arrow

• Combined with another topicalizer, \( \rightarrow \) creates a named alias for the current topic:

```perl
for @lines -> $line is rw {
  s:w/Captain Rimmer!/the bloke/;
  $line _= " who cleans the soup machine!";
  print;
}
#Perl 5
for $line (@lines) {
  $line =~ s/Captain Rimmer!/the bloke/;
  $line .= " who cleans the soup machine!";
  print $line;
}
```
Compare:

```perl
for @lines -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}

$cleanup = -> $line is rw {
    s:w/Captain Rimmer!/the bloke/;
    $line _= " who cleans the soup machine!";
    print;
}
```
• The arrow allows certain non-topicalizers to act as topicalizers:

```perl
if %people{$name}{'details'}{'age'} -> $age {
    print "$age already?\n";
    if ($age > 3000000) {
        print "How was stasis?\n";
    } elsif ($age < 10) {
        print "How 'bout a muffin?\n";
    }
}
```
• This will also work with while:

```perl
while get_next_pattern() -> $pat {
    print grep /<$pat>/, @words;
}
```
This feature isn't useful with all truth tests:

```perl
if $counter > 3 -> $value {
    # do something with $value
}
if $counter > 3 {
    my $value = 1;
    # do something with $value
}
```
Bow and Arrow

• In goofier moments -> is also called "pointy sub".

• So, remember:

  Oh pointy sub, oh pointy, pointy,
  Anoint this variable, anointy, nointy.

  – with apologies to Steve Martin
Methods topicalize their invocant.

```perl
method sub_ether ($self: $message) {
    .transmit( .encode($message) );
}

method sub_ether {
    .transmit( .encoded_message );
}

method sub_ether ( : $message) {
    .transmit( .encode($message) );
}
```
• Subs are not topicalizers.

```perl
sub eddy ($space, $time) {
    print;
}
```

• But using the `is given` property will provide the same behavior.

```perl
sub eddy ($space is given, $time) {
    print;
}
```
Perl Rules!

- Grammar rules topicalize their state object.

```perl
rule lifeform {
    <gelf> | <human> | <mechanic> | <cat>
}
```
The CATCH-er in the Trye

- **CATCH** blocks always topicalize `$!`.

```c
CATCH {
    when Err::WrongUniverse {
        try_new_universe();
    }
}
```
• Bare closures topicalize their first argument.

```perl
%commands = (  
    add => { $^a + $^b },  
    incr => { $_ + 1 },  
);
```
Get Smart... Match

• =~ topicalizes the variable it binds to the match.

```perl
s/Kryten/Holly/;

$name =~ s/Kryten/Holly/;
```
Feeling a Bit Greppish?

- grep-like constructs with a block.
  
  ```perl
  @names = map { chomp; split; } @input;
  ```

- grep-like constructs without a block.
  
  ```perl
  @names = grep /<[A-Z]<alpha>+/, @input;
  ```
Nested Instinct

• Nested topicalizers add some complication.

for @names {
    when /Rimmer/ {
        s/Arnold\s+/\;:
        print;
        print rimmer_quote();
    }
    when /Kryten/ {
        for kryten_quotes() -> $quote {
            print;
        }
    }
}
Nesting Instinct

• There is only one topic at a time.

• Topic obeys the lexical scope of topicalizers.

```plaintext
... when /Kryten/ {
    for kryten_quotes() -> $quote {
        print;
    }
} ...
```
• There is only one topic at a time.

• Topic obeys the lexical scope of topicalizers.

• To keep an outer topic, use a named alias.

```perl
for @names -> $name {
    when /Kryten/ {
        for kryten_quotes() -> $quote {
            print $name;
            print;
        }
    }
}
```
Nested topicalizers within methods obscure `.methodname` calls.

```ruby
method locate ($self, *@characters) {
  .cleanup_names(@characters);
  for @characters -> $name {
    .display_location($name);
  }
  .change_location('Holly');
}
```
Multiple Aliases

- Topicalizers aren't limited to a single alias.

```perl
for @characters -> $role1, $role2, $role3 {
    ...
}

for @humans, @gelfs -> $role1, $role2 {
    ...
}

for @characters; @locations -> $name; $place {
    ...
}
```
Multiple Aliases

• But the topic is consistent in each case.
• There is only one topic.
• The topic is always the first parameter.
Multiple Aliases

- The `is given` (or `is topic`) property may change which parameter becomes the topic.

``` perl
for @characters -> $role1, $role2 is given {
    ...
}
```
First Law of Topic: "Topic is \$\_\$.

Second Law of Topic: There is only one topic.

Third Law of Topic: When in doubt, make a named alias.
The Two Minute Talk

- Isn't that easy?