Skimmable Code

Michael G Schwern
schwern@pobox.com

I didn't get much sleep because I was up all night preparing this talk.

This is because instead of writing the talk Adam Kennedy and I were up talking about EVE Online

Current Location:
ANSONE

Nearest	Stargate (Dunraelare)
Sovereignty	Gallente Federation
Constellation	Jonenor
Region	Sing Laison
Security Level	0.7

Current destination:

System	Dunraelare
Security Level	0.7

BigWhale

PEOPLE & PLACES

SEARCH TYPE: Character
SEARCH STRING
SEARCH

BUDDIES | AGENTS | CORP MEMBERS | BLOCKED | PLACES

LABEL

- Agent Missions
- HQ - Dunraelare VIII - Moon 14
- Ansone Mining Belt - Omber
- Duripant solarsystem
- Goinard Jaspet (Container)
- Dursulaert IX - Federation Customs Testing Facilities (Station)
- Raeghoscon Belt 2 Jaspet 15km
- Tritanium & Pyerite (Trosquesere V - Moon 10)
- Yona - Research Agents

ADD BOOKMARK | CREATE FOLDER

SELECTED ITEM

OVERVIEW

ICI	DISTANCE	TRANSVE	NAME
	18 km		Concord Billboard
	19 km	45 m/s	Nulukizzdan
	10 km		Stargate (Dunraelare)
	21,310 km		Stargate (Trosquesere)

LOCAL [4] | CORP

BigWhale > droni ;)

Tyrus Hawk > a

BigWhale

122 M/S

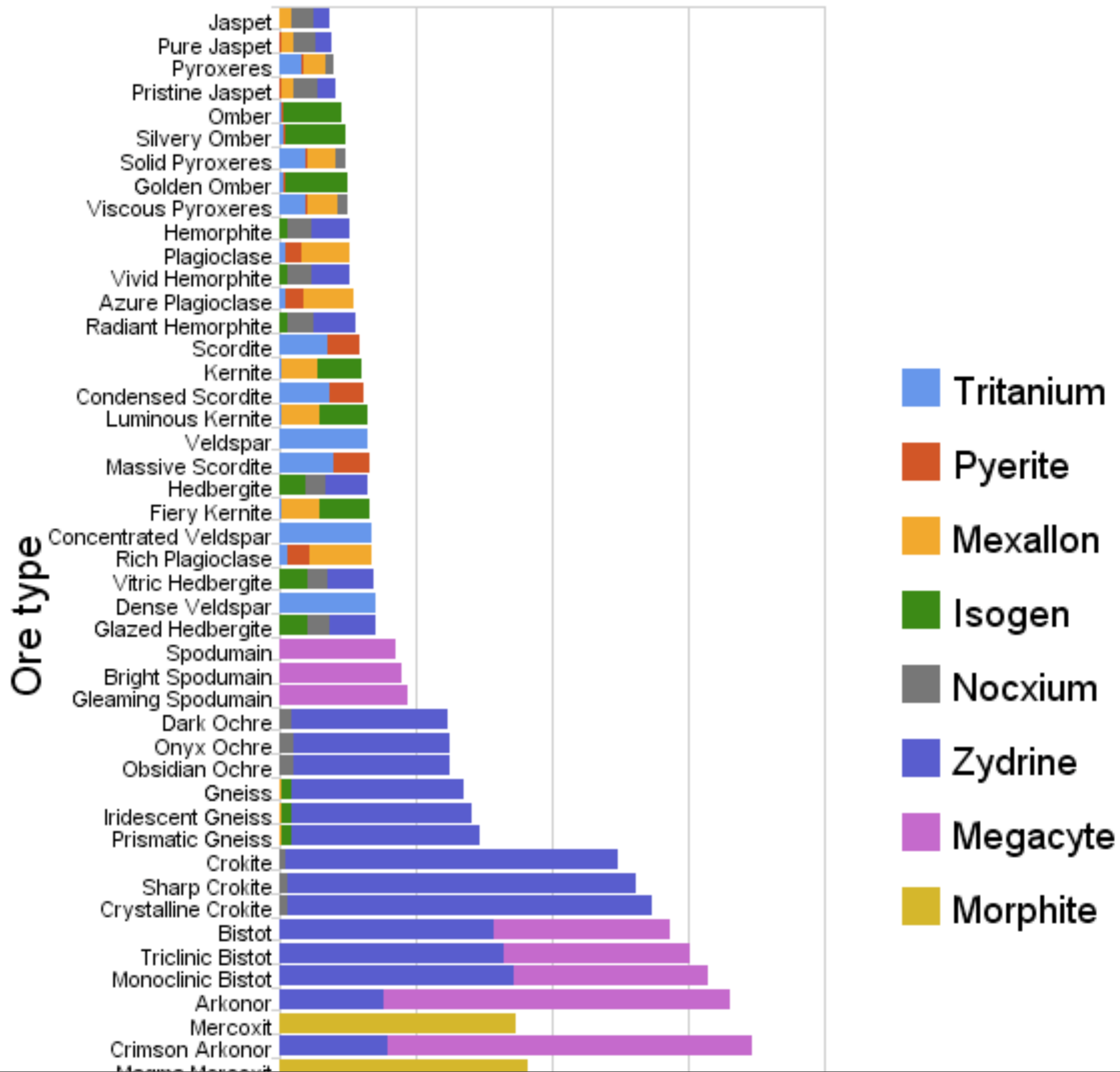
MY CARGO [2]

View

CAPACITY 11,020.7/11,021.0 M³

Golden Omber	Silvery Omber
8729	9639

22:06



We get a little carried away sometimes with the economic side of the game.
Internet Spaceships are Serious Business

Skimmable?

Like when you skim a book.
There's something specific you're looking for.
You want to find it quickly.
Without reading the whole book.

Why Skim?

Focus on a single task

There's a lot of code out there
It would be nice to get some work done without having to read it all

Slice through a hairball

There's a lot of hairy code out there
Work on a single part of it without having to untangle the whole hairball

Applied Best Practices

It's not enough to just say "this is the best way to do it"
It should have a tangible purpose.
Well, here's one.
You're going to see a bunch of BPs in this talk applied.
Hopefully they'll make a little more sense if they didn't already

How do you skim?

Narrow Your Focus

"Work in small chunks"
It's been my chief software engineering message for years
You want to have to remember as little as possible.
Keep as little in your head as possible
Juggle as little as possible
So you can spend less brain on understanding the code
And more on the task at hand
How do you do that?

Lexical Encapsulation

It's mostly done through use of lexical encapsulation
I used this term for a while without realizing that people don't know what it means.

lexical, a.

Pertaining or relating to the words or vocabulary of a language.

Well that's not very useful

```
{
  my $foo = 42;
  print $foo;
}
```

It's the bit between the brackets.
It's the code you can see on the page.
It doesn't go into subroutines.
You can see how this would be useful to narrow focus.

encapsulate, v.

to summarize or isolate as if in a capsule.

Lexical Encapsulation

Using lexical scope to isolate code.
We know how to do this
You put things in subroutines.
Using a subroutine for more than reuse
Using it for encapsulation, isolation, summarizing.

Example

It's good to write skimmable code,
but it's easiest to teach by starting with an example
and making it more skimmable.

Task Branch

Skimming is about narrowing your focus
So make a branch just for whatever task you're doing.

```
svk cp //local/WWW-Mechanize
       //local//WWW-Mechanize-content
```

This will prevent any other changes from interfering.

Make sure it works

Before you start, make sure things already work
If they don't, fix it.
Nothing's work then making a change and having it not work and not knowing if its your change or a bleeding trunk.
How do you do that?

```
$ build test
t/error......ok
t/id.........ok
t/url........ok
All tests successful.
```

Run the tests!

Refactoring?

You've probably heard of refactoring.
It's the process of doing rote transforms on code
changing the design
without changing functionality

1. Refactor
2. Test
3. Commit
4. Repeat
5. ???
6. Profit

In the interest of time, we're going to skip the test and commit parts.
The important thing is you do one refactoring at a time
Test and commit between
This ensures that you haven't broken anything.
And it keeps them from getting tangled up
Small chunks. Less to worry about.

Making code skimmable

I spend a lot of time making code more skimmable through refactoring.
There are two basic refactorings which are immensely useful

Rename

Names are summaries
Summaries of what a variable contains or what a function does.
Good names mean you don't have to see how the variable is initialized
or read the contents of a function
You can just use it
People often use bad names
So in the course of skimming you can make them better

```perl
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $arg = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$arg">/;
        }
        if ( my $arg = delete $parms{format} ) {
            if ($arg eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$arg"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```perl
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $arg = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$arg">/;
        }
        if ( my $arg = delete $parms{format} ) {
            if ($arg eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$arg"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

Extract

Because people write routines which are waaaaay too large
You can often easily move blocks of code out to their own private function
This makes the code far easier to read

```perl
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
if ( my $format = delete $parms{format} ) {
    if ($format eq 'text') {
        require HTML::TreeBuilder;
        my $tree = HTML::TreeBuilder->new();
        $tree->parse($content);
        $tree->eof();
        $tree->elementify(); # just for safety
        $content = $tree->as_text();
        $tree->delete;
    }
    else {
        $self->die( qq{Unknown "format" parameter
"$format"} );
    }
}
```

```
if ( my $format = delete $parms{format} ) {
    if ($format eq 'text') {
        require HTML::TreeBuilder;
        my $tree = HTML::TreeBuilder->new();
        $tree->parse($content);
        $tree->eof();
        $tree->elementify(); # just for safety
        $content = $tree->as_text();
        $tree->delete;
    }
    else {
        $self->die( qq{Unknown "format" parameter
"$format"} );
    }
}
```

```
sub _content_as_text {
    my($self, $content) = @_;

    require HTML::TreeBuilder;
    my $tree = HTML::TreeBuilder->new();
    $tree->parse($content);
    $tree->eof();
    $tree->elementify(); # just for safety
    my $formatted_content = $tree->as_text();
    $tree->delete;

    return $formatted_content;
}
```

What's that "just for safety" thing about?
I dug around in HTML::TreeBuilder a bit, and it's a bit of voodoo that may or may not be necessary. It's not clear.
This is the sort of thing that could really distract you.
LEAVE IT ALONE
It worked before, it'll still work.

Narrow Scope Forgives Many Sins

The weird bit has been isolated.
It's now out of the main line of code you care about.
Put in a ticket about it.
Continue with your task.
Look into it later.
Don't get side-tracked.

```
if ( my $format = delete $parms{format} ) {
    if ($format eq 'text') {
        $content = $self->_content_as_text($content);
    }
    else {
        $self->die( qq{Unknown "format" parameter "$format"} );
    }
}
```

* Much clearer what its doing

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```perl
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                $content = $self->_content_as_text($content);
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                $content = $self->_content_as_text($content);
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub _format_content {
    my($self, $format, $content) = @_;

    if ($format eq 'text') {
        return $self->_content_as_text($content);
    }
    else {
        $self->die( qq{Unknown "format" parameter "$format"} );
    }
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                $content = $self->_content_as_text($content);
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub _check_unhandled_parms {
    my($self, %parms) = @_;

    for my $cmd ( sort keys %parms ) {
        $self->die( qq{Unknown named argument "$cmd"} );
    }
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    } # is HTML

    return $content;
}
```

Hey, that's probably useful in other parts of the code!

Revealing Reuse

* Folks try to write “reusable” code up front
* But they often don’t know how its going to be reused
* Subroutines for encapsulation

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    } # is HTML

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    }

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            if ($format eq 'text') {
                require HTML::TreeBuilder;
                my $tree = HTML::TreeBuilder->new();
                $tree->parse($content);
                $tree->eof();
                $tree->elementify(); # just for safety
                $content = $tree->as_text();
                $tree->delete;
            }
            else {
                $self->die( qq{Unknown "format" parameter "$format"} );
            }
        }
        for my $cmd ( sort keys %parms ) {
            $self->die( qq{Unknown named argument "$cmd"} );
        }
    } # is HTML

    return $content;
}
```

```perl
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    }

    return $content;
}
```

Whitespace

Whitespace in code is like whitespace in any other writing
It makes the structure evident

I reached for a new garrot, let the old one drop into a box on the floor, and began working the wire into a tight coil. "Do you realize, Vlad," said a voice, "that it's been over a year since anyone has tried to kill you?" I looked up. "Do you realize, Krager," I said, "that if you keep walking in here without my seeing you, I'll probably die of a heart attack one of these days and save them the trouble?" He chuckled a little. "No, I mean it, though," he continued. "More than a year. We haven't had any trouble since that punk--What was his name?" "G'ranthar." "Right, G'ranthar. Since he tried to start up a business down on Copper Lane, and you quashed it." "All right," I said, "so things have been quiet. What of it?" "Nothing, really," he said. "It's just that I can't figure out if it's a good sign or a bad sign."

I reached for a new garrot, let the old one drop into a box on the floor, and began working the wire into a tight coil.

"Do you realize, Vlad," said a voice, "that it's been over a year since anyone has tried to kill you?"

I looked up.

"Do you realize, Krager," I said, "that if you keep walking in here without my seeing you, I'll probably die of a heart attack one of these days and save them the trouble?"

He chuckled a little.

"No, I mean it, though," he continued. "More than a year. We haven't had any trouble since that punk--What was his name?"

"G'ranthar."

"Right, G'ranthar. Since he tried to start up a business down on Copper Lane, and you quashed it."

"All right," I said, "so things have been quiet. What of it?"

"Nothing, really," he said. "It's just that I can't figure out if it's a good sign or a bad sign."

This is a conversation, that much is obvious.
You can keep your place better.
Vertical whitespace separates paragraphs
Paragraphs tell you when a new thought is about to begin.
Same goes for code.
Let's talk about CODING STYLES!
That's never produced an argument.

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;
        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    }

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;

        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }
        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    }

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;

        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }

        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }
        $self->_check_unhandled_parms(%parms);
    }

    return $content;
}
```

```
sub content {
    my $self = shift;
    my $content = $self->{content};

    if ( $self->is_html ) {
        my %parms = @_;

        if ( exists $parms{base_href} ) {
            my $base_href = (delete $parms{base_href}) || $self->base;
            $content =~ s/<head>/<head>\n<base href="$base_href">/;
        }

        if ( my $format = delete $parms{format} ) {
            $content = $self->_format_content($format, $content);
        }

        $self->_check_unhandled_parms(%parms);
    }

    return $content;
}
```

That little bit of extra whitespace gives the eyes something more to latch on to.
What about horizontal whitespace?

Similar things
should look similar

```
my %mech_parms = (
    autocheck => 0,
    onwarn => \&WWW::Mechanize::_warn,
    onerror => \&WWW::Mechanize::_die,
    quiet => 0,
    stack_depth => 8675309,
    headers => {},
);
```

```
my %mech_parms = (
    autocheck   => 0,
    onwarn      => \&WWW::Mechanize::_warn,
    onerror     => \&WWW::Mechanize::_die,
    quiet       => 0,
    stack_depth => 8675309,
    headers     => {},
);
```

It's a little thing, but it helps the eye a lot.
Most editors can do it for you.
Don't have to carefully space it out, just use the next tab stop

```
sub _cmp_diag {
    my($self, $got, $type, $expect) = @_;

    $got = defined $got ? "'$got'" : 'undef';
    $expect = defined $expect ? "'$expect'" : 'undef';

    local $Level = $Level + 1;
    return $self->diag(sprintf <<DIAGNOSTIC, $got, $type, $expect);
    %s
        %s
    %s
DIAGNOSTIC
}
```

Example from Test::Builder

```
sub _cmp_diag {
    my($self, $got, $type, $expect) = @_;

    $got    = defined $got    ? "'$got'"    : 'undef';
    $expect = defined $expect ? "'$expect'" : 'undef';

    local $Level = $Level + 1;
    return $self->diag(sprintf <<DIAGNOSTIC, $got, $type, $expect);
    %s
        %s
    %s
DIAGNOSTIC
}
```

Example from Test::Builder
Sometimes, in the process of doing this, you discover something that should be in a routine.

```
my $out = "ok";
$out .= " $self->{Curr_Test}" if $self->use_numbers;
$out .= " # skip";
$out .= " $why" if length $why;
$out .= "\n";
```

```
my $out = "ok";
$out   .= " $self->{Curr_Test}" if $self->use_numbers;
$out   .= " # skip";
$out   .= " $why"               if length $why;
$out   .= "\n";
```

Strings lined up
Conditions lined up

```
my($Testout, $Testerr);
sub _dup_stdhandles {
    my $self = shift;

    $self->_open_testhandles;

    # Set everything to unbuffered else plain prints to STDOUT will
    # come out in the wrong order from our own prints.
    _autoflush($Testout);
    _autoflush(\*STDOUT);
    _autoflush($Testerr);
    _autoflush(\*STDERR);

    $self->output($Testout);
    $self->failure_output($Testerr);
    $self->todo_output($Testout);
}
```

```
my($Testout, $Testerr);
sub _dup_stdhandles {
    my $self = shift;

    $self->_open_testhandles;

    # Set everything to unbuffered else plain prints to STDOUT will
    # come out in the wrong order from our own prints.
    _autoflush($Testout);
    _autoflush(\*STDOUT);
    _autoflush($Testerr);
    _autoflush(\*STDERR);

    $self->output        ($Testout);
    $self->failure_output($Testerr);
    $self->todo_output   ($Testout);
}
```

```
elsif( ($re, $opts) = $regex =~ m{^ /(.*)/ (\w*) $ }sx or
  (undef, $re, $opts) = $regex =~ m{^ m([^\w\s]) (.+) \1 (\w*) $}sx
) {
    $usable_regex = length $opts ? "(?$opts)$re" : $re;
}
```

```
# Check for '/foo/' or 'm,foo,'
elsif( ($re, $opts)        = $regex =~ m{^ /(.*)/              (\w*) $}sx   or
       (undef, $re, $opts) = $regex =~ m{^ m([^\w\s]) (.+) \1 (\w*) $}sx
     )
{
    $usable_regex = length $opts ? "(?$opts)$re" : $re;
}
```

My god, it's almost readable

Different Things Should Look Different

Simplest example of this is...

$lexical

$Global

At a glance you can spot which variables require special attention.
No need to remember that.
If you adopt one naming convention, make it this one.
"Global" also means file scoped lexicals.
So what it really should be is...

$one_subroutine

$Many_Subroutines

Red Flags

Deeply Nested

```
sub fixin {    # stolen from the pink Camel book, more or less
    my ( $self, @files ) = @_;

    my ($does_shbang) = $Config{'sharpbang'} =~ /^\s*\#\!/;
    for my $file (@files) {
        my $file_new = "$file.new";
        my $file_bak = "$file.bak";

        open( my $fixin, '<', $file ) or croak "Can't process '$file': $!";
        local $/ = "\n";
        chomp( my $line = <$fixin> );
        next unless $line =~ s/^\s*\#!\s*//;    # Not a shbang file.
        # Now figure out the interpreter name.
        my ( $cmd, $arg ) = split ' ', $line, 2;
        $cmd =~ s!^.*/!!;

        # Now look (in reverse) for interpreter in absolute PATH (unless perl).
        my $interpreter;
        if ( $cmd eq "perl" ) {
            if ( $Config{startperl} =~ m,^\#!.*/perl, ) {
                $interpreter = $Config{startperl};
                $interpreter =~ s,^\#!,,;
            }
            else {
                $interpreter = $Config{perlpath};
            }
        }
        else {
            my (@absdirs)
                = reverse grep { $self->file_name_is_absolute } $self->path;
            $interpreter = '';

            foreach my $dir (@absdirs) {
                if ( $self->maybe_command($cmd) ) {
                    warn "Ignoring $interpreter in $file\n"
                        if $Verbose && $interpreter;
                    $interpreter = $self->catfile( $dir, $cmd );
                }
            }
        }
        ...
```

It's too long
There's too many conditions
You have to remember too much context.
Oh, and that for loop has a continue block at the end.

Globals

Global variables throw off all skimming.
Sometimes globals are handy
But often you can replace them with a file-scoped lexical.

```
{
    my $Opened_Testhandles = 0;

    sub _open_testhandles {
        my $self = shift;

        return if $Opened_Testhandles;

        # We dup STDOUT and STDERR so people can change them in their
        # test suites while still getting normal test output.
        open( $Testout, ">&STDOUT") or die "Can't dup STDOUT:  $!";
        open( $Testerr, ">&STDERR") or die "Can't dup STDERR:  $!";

        $Opened_Testhandles = 1;

        return 1;
    }
}
```

```
main(@ARGV);

sub main {
    my @args = @_;

    ...
}
```

Subroutines should have one way in, via arguments
And one way out, via return().
Globals throw this off.
So pass in the global.
I like this technique when writing scripts because it means you can test main() like any other function.

Long routines

How long is too long?
How much do you have to remember?
Does it all fit on one page? (Your page size will vary)
Is what's not on the page easily accessible? Via docs or a clever IDE?

Scope closing comments

```
while( $foo > 42 ) {
    ...
    ...
    ...
} # while $foo > 42
```

This means you can't see the whole block at once.
It's too long.

Stay Focused

Get In

Do your task

Get out

Separate branch
Push any oddities you find off to the tracker.

Thanks

Questions?