wherevs

- `qw()` may not always create a list so wrap it in parens

```
use Linode::CLI::Util qw(:basic :json);
```

Becomes:

```
use Linode::CLI::Util (qw(:basic :json));
```

- don't use `qw()` for only one arg

```
use String::Random qw(random_string);
```

Becomes:

```
use String::Random 'random_string';
```

- `\t`s in code cause unwed teen moms
  - if you are in to teen pregnancy, for the love of all that is holy don't use tabs to align multi line assignments, use spaces

scripts overall

- move into `bin` sub-directory
- eliminate pl extension
- make `linode-linode` and `linode-domain` symlink to `linode` switch based on `$0`
  - I'm unsure on how repo looks for this one. After install it should just be a symlink, in the repo/tarball it should either be a symlink or not exist
- move even more out of the scripts and into modules
  - one or two statements at most
  - code lives in `Linode/CLI/Command/Domain.pm`, `Linode/CLI/Command/Linode.pm` and `Linode/CLI/Command.pm`

In both `linode-linode` and `linode-domain`

- don't predeclare/scope variables when not ncessary

```
my $cmdargs = {};
$cmdargs = Linode::CLI::Util::eat_cmdargs($mode);
```

Becomes:

```
my $cmdargs = Linode::CLI::Util::eat_cmdargs($mode);
```

---

linode.pl

- use `FindBin` instead of `BEGIN` block:

```
BEGIN {
    use File::Basename;
    use File::Spec::Functions qw(rel2abs);
    my $base_dir = dirname(rel2abs($0));
    push @INC, "$base_dir";
}
...
system(dirname(rel2abs($0)) . "/linode-$object.pl", @ARGV);
```

Becomes:

```
use FindBin;
use lib "$FindBin::Bin/../lib";
...
system( $FindBin::Bin . '/linode-$object.pl', @ARGV);
```

---

packages overall

- when collecting named params don't use delete unless you are going to validate what is left over

```
my $api_obj      = delete $params{api_obj};
my $object_list  = delete $params{object_list};
my $field_list   = delete $params{field_list};
```

Becomes:

```
my $api_obj      = $params{api_obj};
my $object_list  = $params{object_list};
my $field_list   = $params{field_list};
```

- throw stuff straight into `$self` skipping intermediary assignments unless you are going to pre-process it further

```
my $api_obj      = $params{api_obj};
my $object_list  = $params{object_list};
my $field_list   = $params{field_list};
...
$self->{_api_obj} = $api_obj;
$self->{_field_list} = $field_list;
$self->{_output_fields} = $output_fields;
```

Becomes:

```
$self->{_api_obj}        = $params{api_obj};
$self->{_field_list}     = $params{object_list};
$self->{_output_fields}  = $params{field_list};
```

- avoid multiple shifts

```
my $self = shift;
my $label = shift || 0;
```

Becomes:

```
my ($self, $label) = @_;
$label ||= 0;
```

- avoid interpolation where not necessary

```
return "Not implemented.";
# or
return "$return" . ('=' x 72) . "\n";
```

Becomes:

```
return 'Not implemented.';
return $return . ('=' x 72) . "\n";
```

- sometimes undef is okay, if it doesn't cause warnings

```
    my $label = shift || 0;
    ...
        next if ($label && $object_label ne $label);
```

Becomes:

```
    my $label = shift || 0;
    ...
        next if $object_label ne $label;

    # OR
    my $label = shift;
    ...
        next if ($label && $object_label ne $label);
```

I personally prefer the 2nd, especially when combined with the next item.

- unless needful, leave args wherever we put them

```
    my ($self, %params) = @_;
    my $label = delete $params{label} || 0;
    my $output_format = delete $params{output_format} || 'human';
    ...
        next if ($label && $object_label ne $label);
    ...
    if ($output_format eq 'raw') {
```

Becomes:

```
    my ($self, %params) = @_;
    $params{output_format} ||= 'human';
    ...
        next if ($params{label} && $object_label ne $params{label});
    ...
    if ($output_format eq 'raw') {
```

- use ternaries in places that cry for c style switch

```
    if ($output_format eq 'raw') {
        return $out_hashref;
    }
    elsif ($output_format eq 'json') {
        return json_response($out_hashref);
    }
    else {
        return "Not implemented.";
    }
```

Becomes:

```
    return $params{output_format} eq 'raw'  ? $out_hashref :
           $params{output_format} eq 'json' ? json_response($output_format) :
           'Not implemented.';
```

```
* side note: `Linode::CLI::Object::list` sets default output format to `human`
```

- consistent arg passing
  - some methods take a list some, named args (eg `_poll_and_wait`)
- try really hard to keep things under 80 columns
  - it's okay to occasionally go over
  - `args` is shorter than `params` and just as descriptive
  - avoid superfluous stuffs in var names `api` instead of `api_obj`, `list` instead of `object_list`, `stackscripts` instead of `stackscript_list`, etc

```
    our @EXPORT_OK
        = qw( json_response error load_config human_displaymemory %humanstatus
```

Becomes:

```
    our @EXPORT_OK = qw( json_response error load_config human_displaymemory
        %humanstatus %humanyn $cli_err @MODES );
```

Combing stuff from prior and this one gives:

```
sub new_from_list {
    my ($class, %params) = @_;

    my $api_obj             = delete $params{api_obj};
    my $stackscript_list    = delete $params{stackscript_list} || $api_obj->st

    return $class->SUPER::new_from_list(
        api_obj     => $api_obj,
        object_list => $stackscript_list,
    );
}
```

Becomes:

```
sub new_from_list {
    my ( $class, %args ) = @_;
    $args{stackscripts} ||= $args{api}->stackscripts;
    return $class->SUPER::new_from_list(%args);
}
```