

Constructing Command Line Applications in Perl

Jason A. Crome

The Command Line? Really?

- Yes, really!
- Easy and fast to manage
- Easy to automate
- Not always one-off scripts, often apps in their own right
- We'll learn everything from quick-and-dirty to full-blown CLI application

Let's Go!

Roll Your Own!

- TIMTOWTDI!
- Just like building your own template engine...
- What could go wrong???

blort, version 1

```
#!/usr/bin/env perl

use v5.26;

my $verbose = 0;
my $force   = 0;

foreach my $opt( @ARGV ) {
    if( $opt eq "-h" ) {
        say help();
        exit 0;
    } elsif( $opt eq "-f" ) {
        $force = 1;
    } elsif( $opt eq "-v" ) {
        $verbose = 1;
    } else {
        say "Invalid option: $opt";
        exit 1;
    }
}

exit blort();

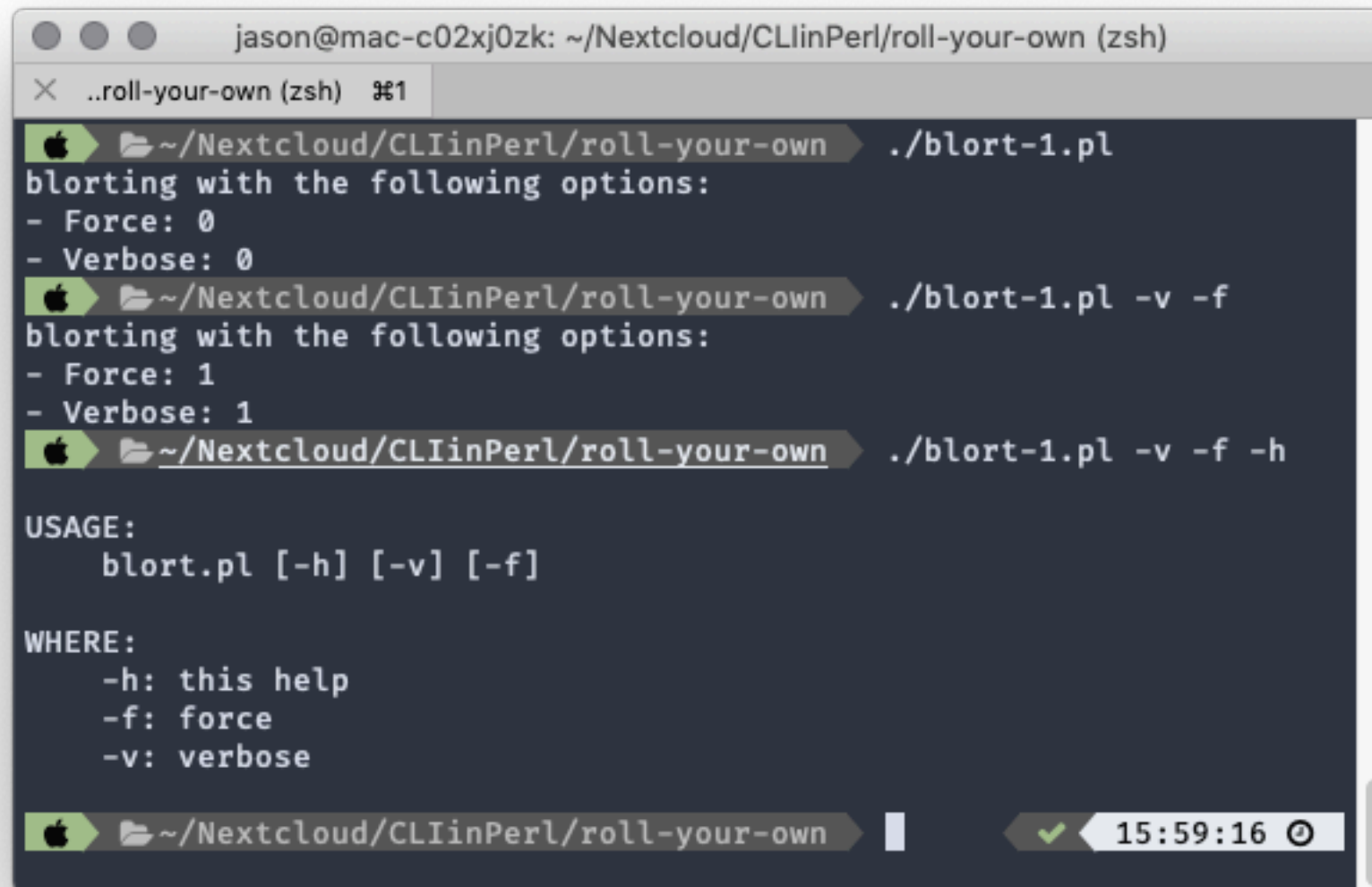
sub blort {
    say "blorting with the following options:";
    say "- Force: $force";
    say "- Verbose: $verbose";

    return 0;
}

sub help {
    return qq{
USAGE:
    blort.pl [-h] [-v] [-f]

WHERE:
    -h: this help
    -f: force
    -v: verbose
    };
}
```

blort, version 1



```
jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/roll-your-own (zsh)
X ..roll-your-own (zsh)  #1
Apple ~/Nextcloud/CLIinPerl/roll-your-own ./blort-1.pl
blorting with the following options:
- Force: 0
- Verbose: 0
Apple ~/Nextcloud/CLIinPerl/roll-your-own ./blort-1.pl -v -f
blorting with the following options:
- Force: 1
- Verbose: 1
Apple ~/Nextcloud/CLIinPerl/roll-your-own ./blort-1.pl -v -f -h

USAGE:
  blort.pl [-h] [-v] [-f]

WHERE:
  -h: this help
  -f: force
  -v: verbose

Apple ~/Nextcloud/CLIinPerl/roll-your-own 15:59:16
```

Let's Upset the Apple Cart!

- Let's expand this to accept short and long arguments
- Once we get that right, let's accept arguments that accept arguments (i.e., `blort.pl -i 5 -v -f`)

blort, version 2

```
#!/usr/bin/env perl

use v5.26;

my $verbose = 0;
my $force   = 0;

foreach my $opt( @ARGV ) {
    if( $opt eq "-h" or $opt eq "--help" ) {
        say help();
        exit 0;
    } elsif( $opt =~ /^-f|--force$/ ) {
        $force = 1;
    } elsif( $opt =~ /^-v|--verbose$/ ) {
        $verbose = 1;
    } else {
        say "Invalid option: $opt";
        exit 1;
    }
}

exit blort();

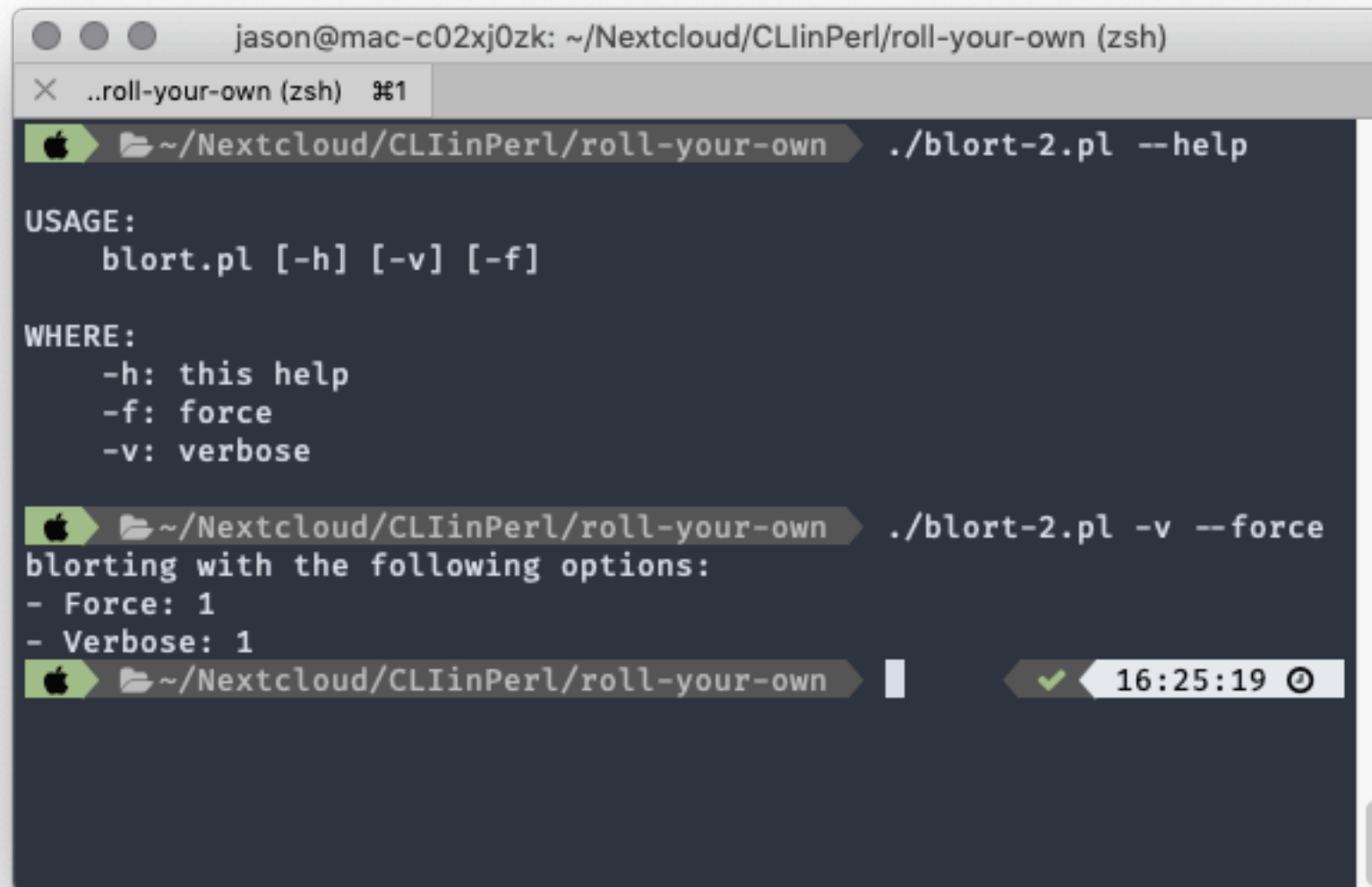
sub blort {
    say "blorting with the following options:";
    say "- Force: $force";
    say "- Verbose: $verbose";

    return 0;
}

sub help {
    return qq{
USAGE:
    blort.pl [-h | --help] [-v | --verbose] [-f | --force]

WHERE:
    -h / --help: this help
    -f / --force: force
    -v / --verbose: verbose
    };
}
```


blort, version 2



```
jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/roll-your-own (zsh)
X ..roll-your-own (zsh) #1
~/Nextcloud/CLIinPerl/roll-your-own ./blort-2.pl --help
USAGE:
  blort.pl [-h] [-v] [-f]
WHERE:
  -h: this help
  -f: force
  -v: verbose
~/Nextcloud/CLIinPerl/roll-your-own ./blort-2.pl -v --force
blorting with the following options:
- Force: 1
- Verbose: 1
~/Nextcloud/CLIinPerl/roll-your-own [ ] ✓ 16:25:19 ⓘ
```

blort, version 3

```
#!/usr/bin/env perl

use v5.26;

my $verbose = 0;
my $force   = 0;
my $iters   = 0;

foreach my $opt( @ARGV ) {
    if( $opt =~ /^-h|--help$/ ) {
        say help();
        exit 0;
    } elsif( $opt =~ /^-f|--force$/ ) {
        $force = 1;
    } elsif( $opt =~ /^-v|--verbose$/ ) {
        $verbose = 1;
    } elsif( $opt =~ /^-i|--iters$/ ) {
        $iters = pop @ARGV;
        say "iters is not a decimal!" unless $iters =~ /\d+$/;
    } else {
        say "Invalid option: $opt";
        exit 1;
    }
}

exit blort();

sub blort {
    say "blorting with the following options:";
    say "- Force: $force";
    say "- Verbose: $verbose";
    say "- Iters: $iters";

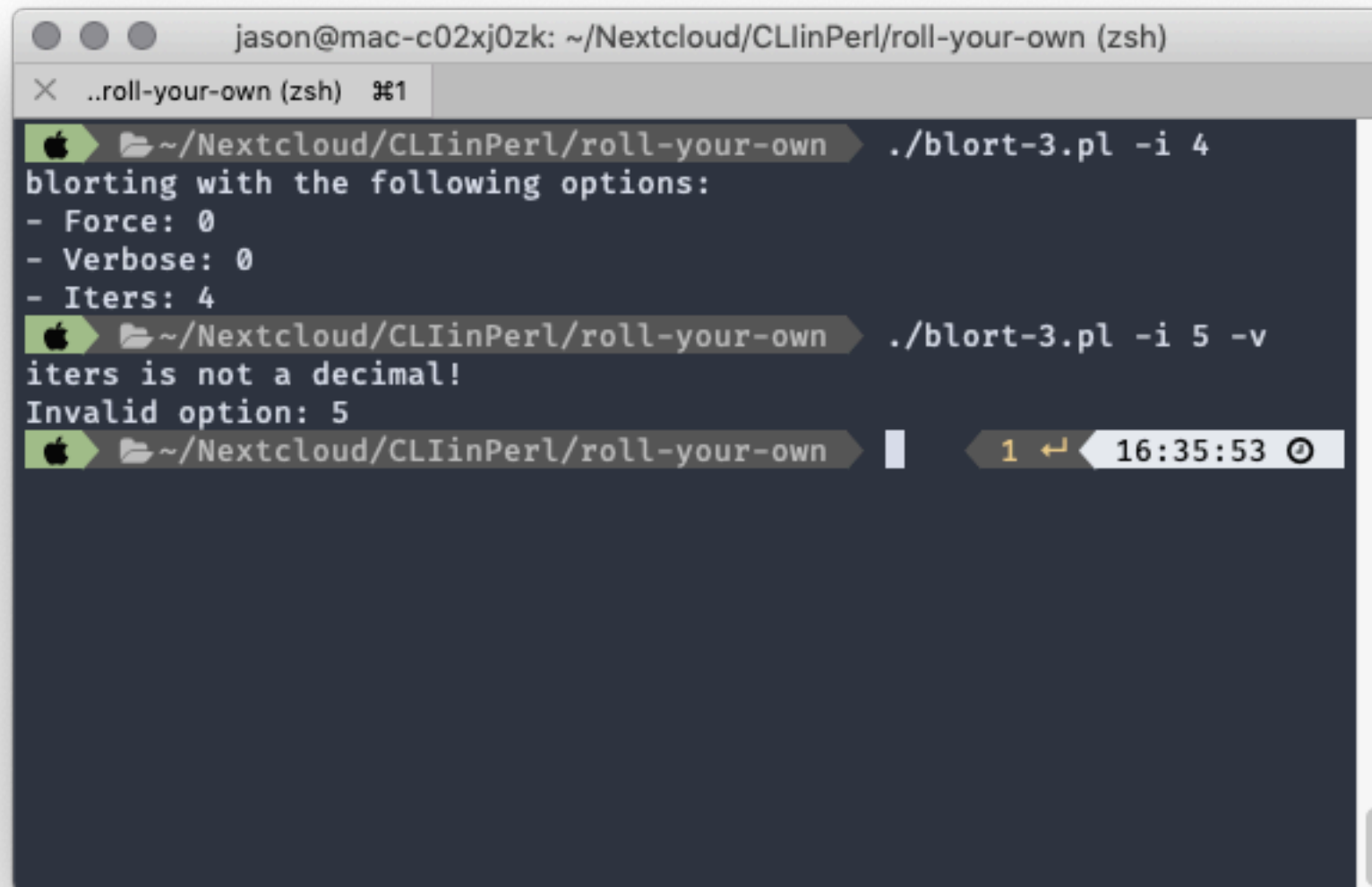
    return 0;
}

sub help {
    return qq{
USAGE:
  blort.pl [-h | --help] [-v | --verbose] [-f | --force] [-i | --iters #]

WHERE:
  -h / --help: this help
  -f / --force: force
  -v / --verbose: verbose
  -i # / --iters #: number of iterations
    };
}
```

Copyright 2019, Jason A. Crome

blort, version 3

A screenshot of a macOS terminal window. The window title is 'jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/roll-your-own (zsh)'. The terminal shows three command-line interactions. The first command is './blort-3.pl -i 4', which outputs 'blorting with the following options:' followed by a list of options: '- Force: 0', '- Verbose: 0', and '- Iters: 4'. The second command is './blort-3.pl -i 5 -v', which outputs an error message: 'iters is not a decimal!' followed by 'Invalid option: 5'. The third command is a partial './blort-3.pl' which is followed by a cursor. The terminal interface includes a dark background, a light-colored prompt, and a status bar at the bottom right showing '1' and a time of '16:35:53'.

Problems!

- It doesn't read well
- It doesn't maintain well
- It's not easy to expand
- Frankly, it just sucks!

Getopt::Long

- The old tried-and-true
- It's practically part of Perl core
- Fast and flexible
- Parses arguments only
- Most other CLI argument parsers build on this

blort, version 4

```
#!/usr/bin/env perl

use v5.26;
use Getopt::Long;
use Pod::Usage;

my $verbose = 0;
my $force   = 0;
my $iters   = 0;

GetOptions(
    'v|verbose' => \$verbose,
    'f|force'   => \$force,
    'i|iters=i' => \$iters,
    'h|help'     => sub{ pod2usage(1); },
) or pod2usage(2);

exit blort();

sub blort {
    say "blorting with the following options:";
    say "- Force: $force";
    say "- Verbose: $verbose";
    say "- Iters: $iters";

    return 0;
}
```

blort, version 4

=pod

=head1 NAME

blort-4.pl - All your blort are belong to us!

=head1 SYNOPSIS

```
blort.pl [-h | --help] [-v | --verbose] [-f | --force] [-i | --iters #]
```

=head1 DESCRIPTION

F<blort-4.pl> is for all your blorting needs.

=head1 OPTIONS

=over 4

=item * -h / --help

Get some help.

=item * -f / --force

Forcibly blort.

=item * -v / --verbose

Verbosely blort.

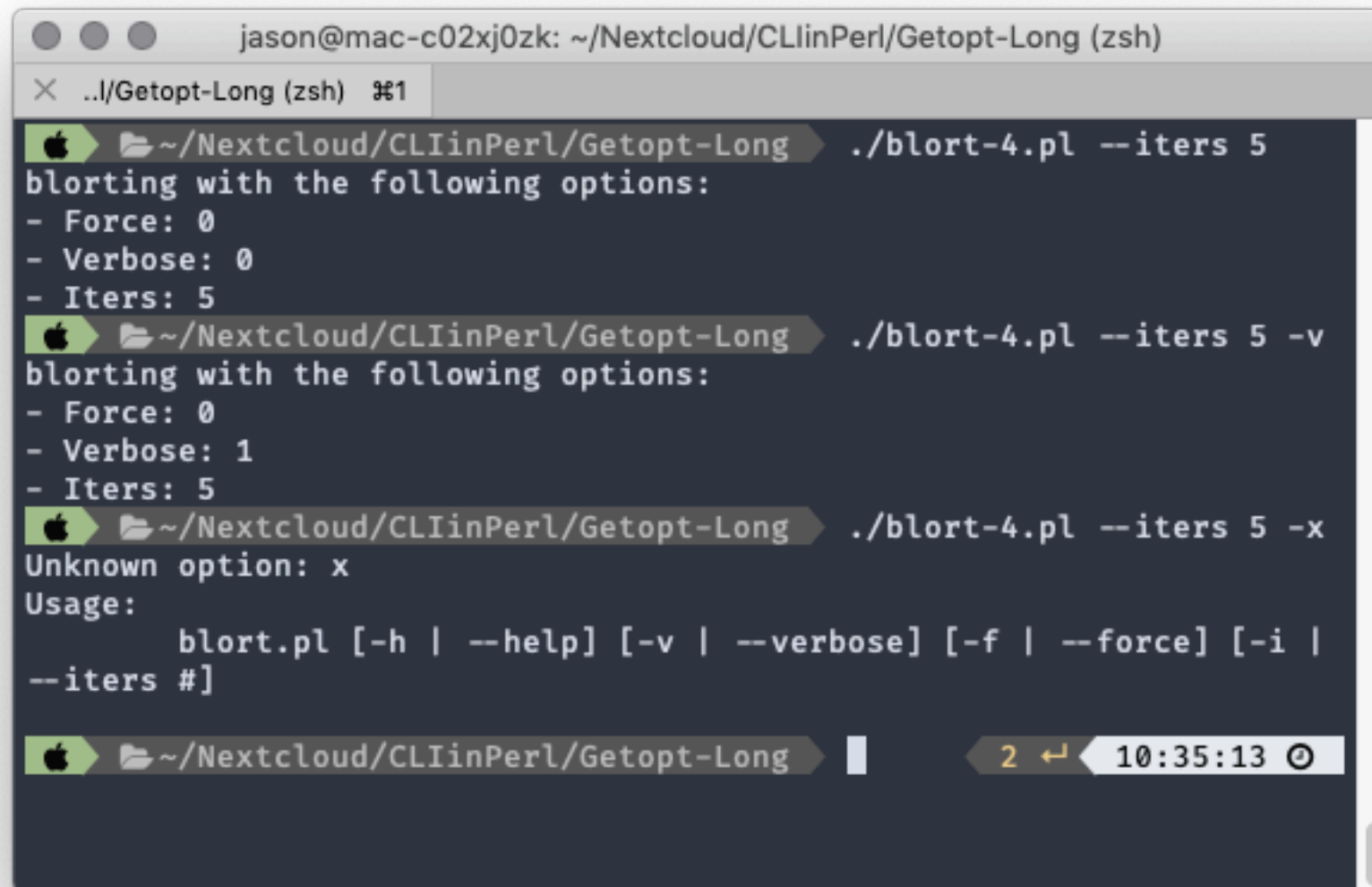
=item * -i / --iters

How many times should we blort?

=back

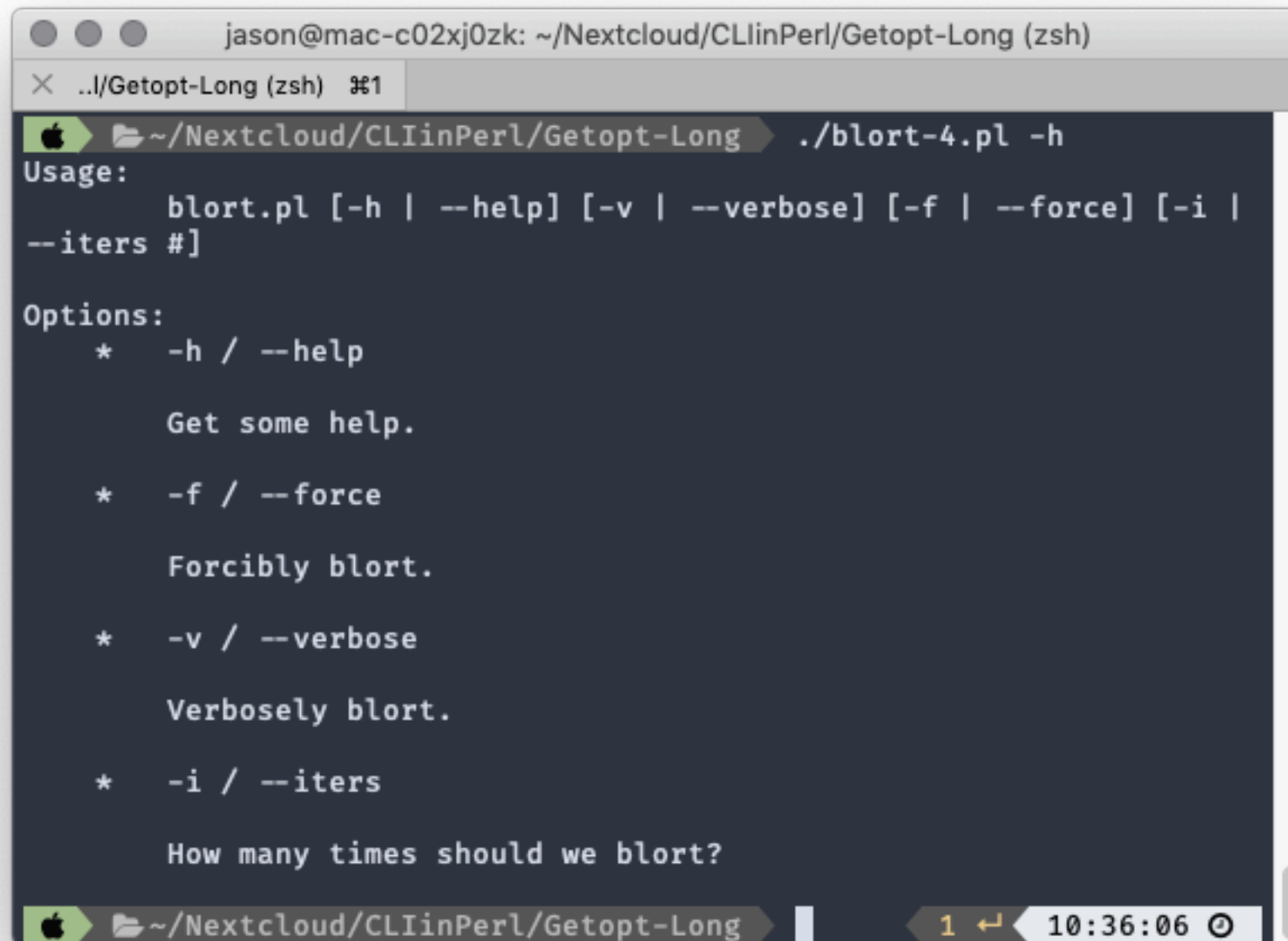
=cut

blort, version 4

A terminal window titled 'jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/Getopt-Long (zsh)'. The window shows three command-line interactions with the 'blort-4.pl' script. The first command is './blort-4.pl --iters 5', which outputs 'blorting with the following options:' followed by '- Force: 0', '- Verbose: 0', and '- Iters: 5'. The second command is './blort-4.pl --iters 5 -v', which outputs 'blorting with the following options:' followed by '- Force: 0', '- Verbose: 1', and '- Iters: 5'. The third command is './blort-4.pl --iters 5 -x', which outputs 'Unknown option: x' and then 'Usage: blort.pl [-h | --help] [-v | --verbose] [-f | --force] [-i | --iters #]'. The terminal window has a dark background and a light-colored title bar. The command prompt is a green arrow pointing to a folder icon, followed by the path '~/Nextcloud/CLIinPerl/Getopt-Long'. The output is in a monospaced font. The status bar at the bottom shows a cursor, a line number '2', a back arrow, and a timestamp '10:35:13' with a refresh icon.

```
jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/Getopt-Long (zsh)
X ../Getopt-Long (zsh) #1
Apple ~/Nextcloud/CLIinPerl/Getopt-Long ./blort-4.pl --iters 5
blorting with the following options:
- Force: 0
- Verbose: 0
- Iters: 5
Apple ~/Nextcloud/CLIinPerl/Getopt-Long ./blort-4.pl --iters 5 -v
blorting with the following options:
- Force: 0
- Verbose: 1
- Iters: 5
Apple ~/Nextcloud/CLIinPerl/Getopt-Long ./blort-4.pl --iters 5 -x
Unknown option: x
Usage:
      blort.pl [-h | --help] [-v | --verbose] [-f | --force] [-i |
--iters #]
Apple ~/Nextcloud/CLIinPerl/Getopt-Long 2 ↩ 10:35:13 ⌂
```


blort, version 4



A terminal window titled "jason@mac-c02xj0zk: ~/Nextcloud/CLlinPerl/Getopt-Long (zsh)". The window shows the command `./blort-4.pl -h` being executed. The output displays the usage and options for the `blort.pl` script. The usage line is `blort.pl [-h | --help] [-v | --verbose] [-f | --force] [-i | --iters #]`. The options section lists five flags: `-h / --help` (Get some help.), `-f / --force` (Forcibly blort.), `-v / --verbose` (Verbosely blort.), and `-i / --iters` (How many times should we blort?). The terminal window has a dark background and a light-colored title bar. The bottom status bar shows the current directory, a cursor, and the time 10:36:06.

```
jason@mac-c02xj0zk: ~/Nextcloud/CLlinPerl/Getopt-Long (zsh)
X ..l/Getopt-Long (zsh) #1
~/Nextcloud/CLlinPerl/Getopt-Long ./blort-4.pl -h
Usage:
    blort.pl [-h | --help] [-v | --verbose] [-f | --force] [-i |
--iters #]
Options:
    *   -h / --help
        Get some help.
    *   -f / --force
        Forcibly blort.
    *   -v / --verbose
        Verbosely blort.
    *   -i / --iters
        How many times should we blort?
```

More fun with Getopt::Long

- Other argument types args
- Inline variable declaration
- My hack
- Multiple occurrences of same argument
- Comma separated options

blort, version 5

```
#!/usr/bin/env perl

use v5.26;
use Getopt::Long;
use Pod::Usage;

GetOptions(
    'v|verbose' => \my $verbose, # declare var inline
    'f|force'   => \my $force,
    'i|iters=i' => \my $iters,
    'd|devs=s'  => \my @devs,    # no default value!
    'h|help'    => sub{ pod2usage(1); },
) or pod2usage(2);

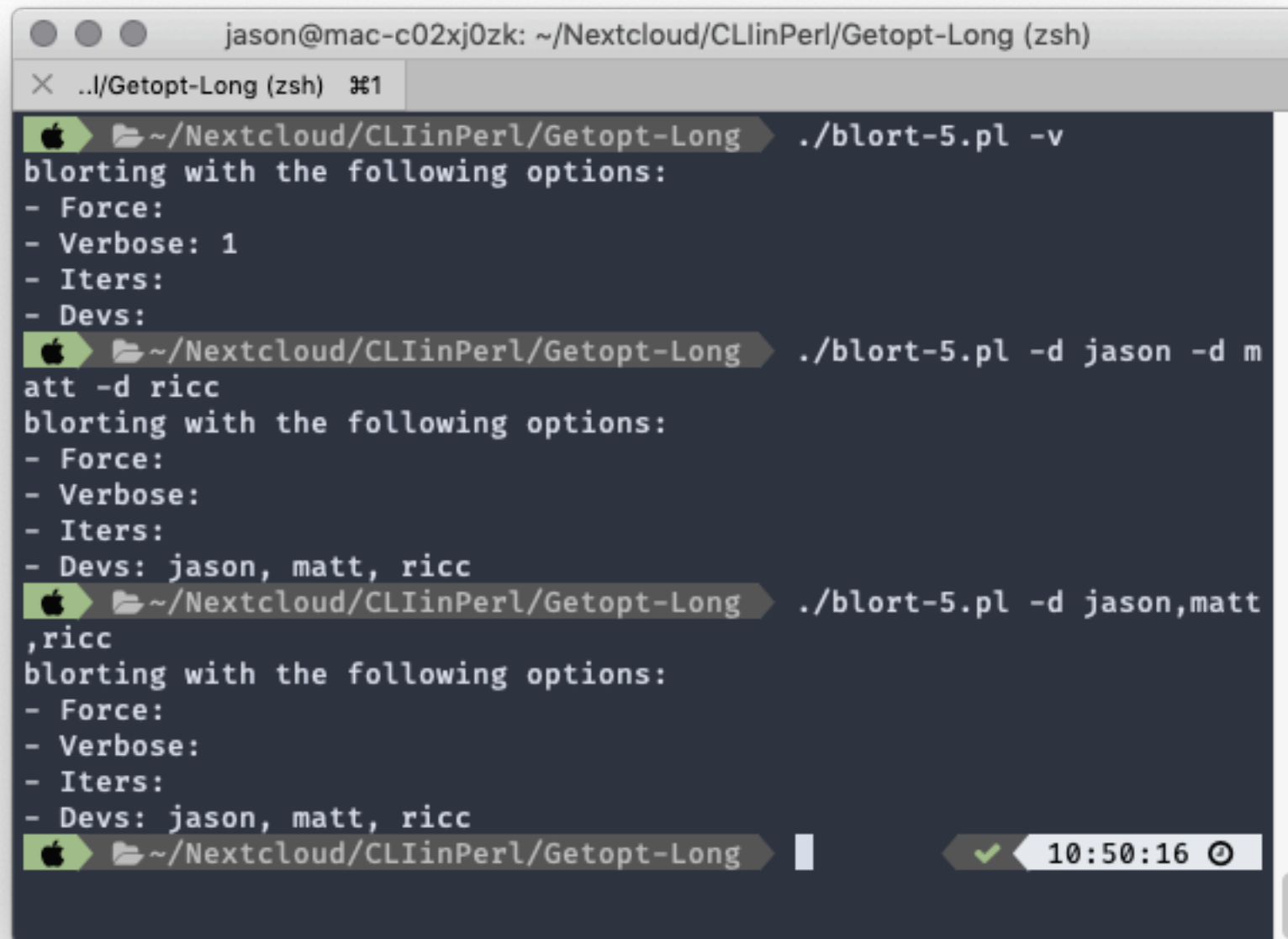
exit blort();

sub blort {
    @devs = split( /\./, join( ',', @devs ) );

    say "blorting with the following options:";
    say "- Force: $force";
    say "- Verbose: $verbose";
    say "- Iters: $iters";
    say "- Devs: ' . join( ',', @devs );

    return 0;
}
```

blort, version 5



A terminal window titled 'jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/Getopt-Long (zsh)' with a sub-tab ' ../Getopt-Long (zsh) #1'. The terminal shows three invocations of the script `./blort-5.pl`. Each invocation is preceded by a prompt showing the current directory as `~/Nextcloud/CLIinPerl/Getopt-Long`. The first invocation uses `-v` and lists options: Force, Verbose: 1, ITERS, and Devs. The second invocation uses `-d jason -d matt -d ricc` and lists options: Force, Verbose, ITERS, and Devs: jason, matt, ricc. The third invocation uses `-d jason,matt,ricc` and lists the same options. At the bottom right, there is a status bar with a green checkmark, a white bar, the time '10:50:16', and a refresh icon.

```
jason@mac-c02xj0zk: ~/Nextcloud/CLIinPerl/Getopt-Long (zsh)
X ../Getopt-Long (zsh) #1
~/Nextcloud/CLIinPerl/Getopt-Long ./blort-5.pl -v
blorting with the following options:
- Force:
- Verbose: 1
- ITERS:
- Devs:
~/Nextcloud/CLIinPerl/Getopt-Long ./blort-5.pl -d jason -d m
att -d ricc
blorting with the following options:
- Force:
- Verbose:
- ITERS:
- Devs: jason, matt, ricc
~/Nextcloud/CLIinPerl/Getopt-Long ./blort-5.pl -d jason,matt
,ricc
blorting with the following options:
- Force:
- Verbose:
- ITERS:
- Devs: jason, matt, ricc
~/Nextcloud/CLIinPerl/Getopt-Long [ ] [✓] 10:50:16 [refresh]
```

Getopt::Long Criticisms

- A bit tricky to get right for newcomers
- Don't forget to check return value of `GetOptions()`.

Now, with more Moose!



MooseX::App

- Build CLI like you would an app
- Makes it easy to create well structured, documented code
- If you know Moose, you know MooseX::App
- Anything you do with Moose, you can do in MooseX::App
- If your using code that uses Moose, using this costs you nothing extra

MooseX::App

- Build a base class with common functionality
- Create roles that contain discrete units of functionality
- Create command classes that use roles, add new functionality

blort, version 6 (Base Class)

```
package Blort;

use v5.26;
use MooseX::App qw( Color Term );
with qw(
    Blort::Role::WithDebug
    Blort::Role::WithVerbose
    Blort::Role::WithTerm
);
use feature 'signatures';
no warnings 'experimental::signatures';

option target => (
    is          => 'ro',
    isa         => 'Str',
    lazy        => 1,
    default     => 'test',
    documentation => 'Target environment',
);

# NOTE TO SELF: cmd_env specifies an env var that can be set instead of a parameter

# Display a message in verbose mode
sub _say( $self, $message ) {
    return unless $self->verbose;
    say $message;
}

# Display a message in debug mode
sub _say_debug ( $self, $message ) {
    return unless $self->debug;
    say $message;
}

# Translate package name from camel case into hyphenated name
app_command_name {
    my @parts = split( /[_\s]+|\b(?:[A-Z])(?=[A-Z])|(?:[A-Z])(?=[A-Z][a-z])/ , shift );
    return lc(join('-',@parts));
};

app_namespace 'Blort::Commands';

sub BUILD {
    my $self = shift;
    $self->_say_debug( "Using Target: " . $self->target );
}

1;
```

blort, version 6 (Instance Script)

```
#!/usr/bin/env perl
```

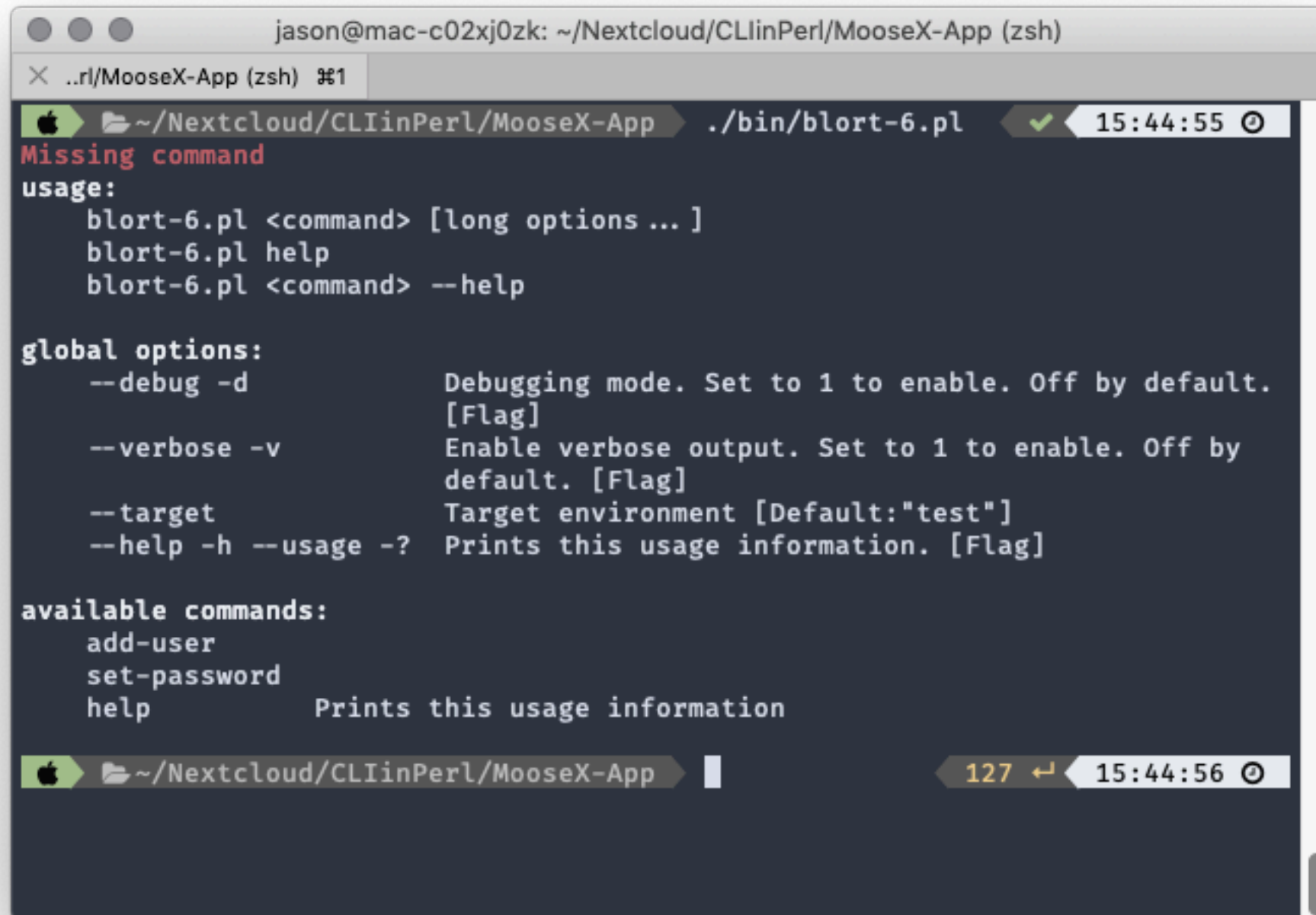
```
use lib './lib';
```

```
use v5.26;
```

```
use Blort;
```

```
Blort→new_with_command→run;
```

blort, version 6



A terminal window titled 'jason@mac-c02xj0zk: ~/Nextcloud/CLInPerl/MooseX-App (zsh)'. The window shows the execution of './bin/blort-6.pl' which results in a 'Missing command' error. The program then displays its usage and global options. The usage section shows 'blort-6.pl <command> [long options ...]', 'blort-6.pl help', and 'blort-6.pl <command> --help'. The global options section lists '--debug -d' (Debugging mode), '--verbose -v' (Enable verbose output), '--target' (Target environment), and '--help -h --usage -?' (Prints this usage information). The available commands section lists 'add-user', 'set-password', and 'help' (Prints this usage information). The terminal window has a dark background with light text. The top bar shows the file path and the command executed. The bottom bar shows the exit code '127' and the time '15:44:56'.

```
jason@mac-c02xj0zk: ~/Nextcloud/CLInPerl/MooseX-App (zsh)
X ..rl/MooseX-App (zsh) #1
~/Nextcloud/CLInPerl/MooseX-App ./bin/blort-6.pl 15:44:55
Missing command
usage:
  blort-6.pl <command> [long options ... ]
  blort-6.pl help
  blort-6.pl <command> --help

global options:
  --debug -d          Debugging mode. Set to 1 to enable. Off by default.
                      [Flag]
  --verbose -v        Enable verbose output. Set to 1 to enable. Off by
                      default. [Flag]
  --target            Target environment [Default:"test"]
  --help -h --usage -? Prints this usage information. [Flag]

available commands:
  add-user
  set-password
  help              Prints this usage information

~/Nextcloud/CLInPerl/MooseX-App 127 15:44:56
```

blort, version 6 (WithPassword)

```
package Blort::Role::WithPassword;

use v5.26;
use MooseX::App::Role;
use feature 'signatures';
no warnings 'experimental::signatures';

option 'password' => (
    is          => 'rw',
    isa         => 'Str',
    required    => 1,
    documentation => 'Password for the new user',
    cmd_aliases => [qw(p)],
);

1;
```

blort, version 6 (AddUser Command)

```
package Blort::Commands::AddUser;

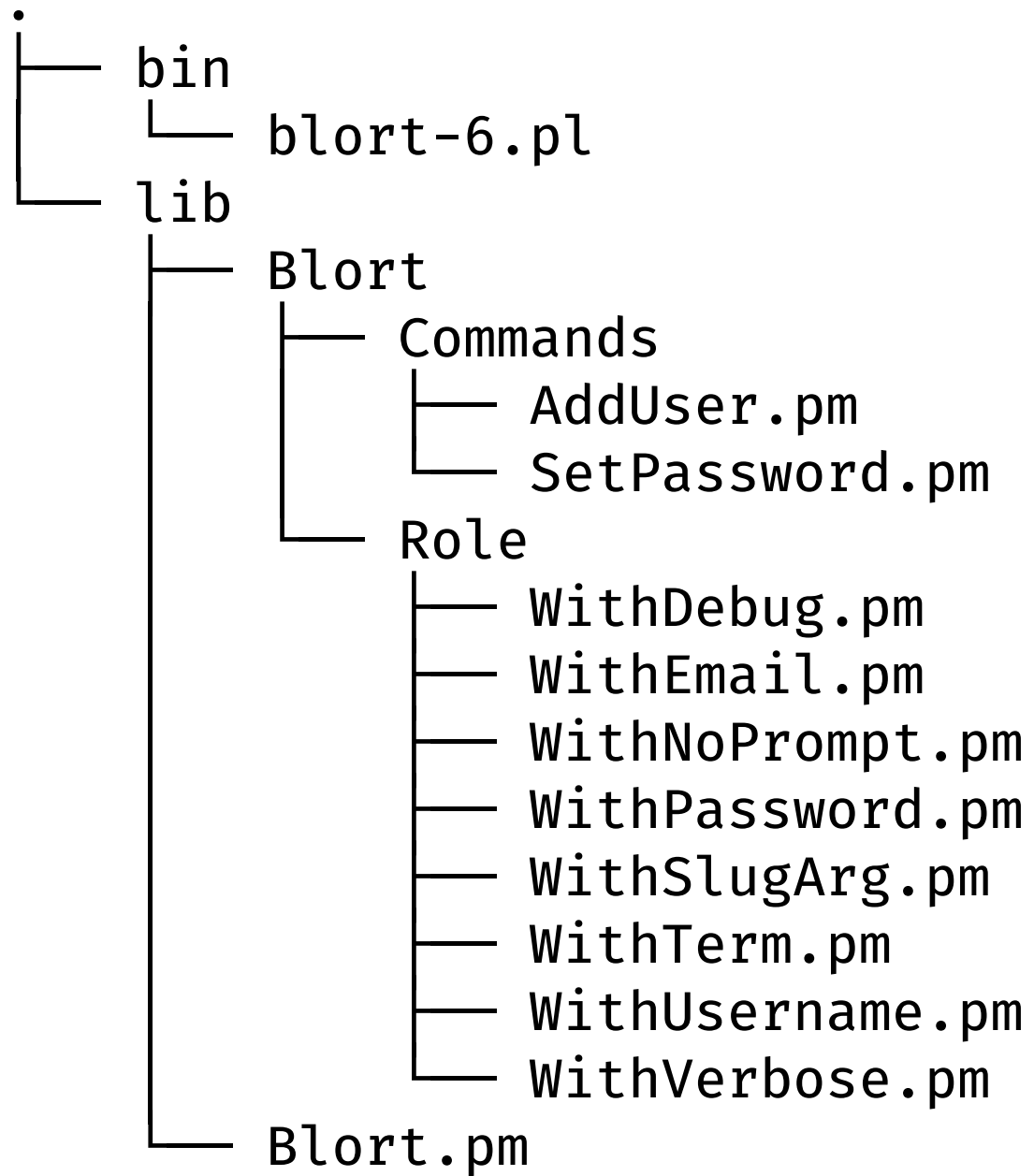
use v5.26;
use MooseX::App::Command;
extends qw( Blort );
with qw(
    Blort::Role::WithUsername
    Blort::Role::WithPassword
    Blort::Role::WithEmail
);
use feature 'signatures';
no warnings 'experimental::signatures';

option 'godmode' => (
    is          => 'rw',
    isa         => 'Bool',
    required    => 0,
    documentation => 'Enable Doom-like God-mode for this user?',
    cmd_aliases => [qw(g)],
    default     => 0,
);

sub run ( $self ) {
    my $user = $self->username;
    $self->_say_debug( "Running AddUser ... " );
    $self->_say( "Creating $user ... done!" );
}

1;
```

blort, version 6 (Project Tree)



5 directories, 12 files

MooseX::App Drawbacks

- S - L - O - W - ! - ! - !
- Maximum overkill for many situations





The Middle Ground

App::Cmd

- Older OO-framework for CLI apps
- Predates Moose
- Many of the same benefits, but lightweight
- Nice compromise between speed and functionality
- Stable and proven

App::Cmd Base Class

```
package YourApp;  
use App::Cmd::Setup -app;  
1;
```

App::Cmd Instance Script

```
use YourApp;  
YourApp->run;
```

App::Cmd Command Class

```
package YourApp::Command::blort;
use YourApp -command;
use strict; use warnings;

sub abstract { "blortex algorithm" }

sub description { "Long description on blortex algorithm" }

sub opt_spec {
    return (
        [ "blortex|x", "use the blortex algorithm" ],
        [ "recheck|r", "recheck all results" ],
    );
}

sub validate_args {
    my ($self, $opt, $args) = @_;

    # no args allowed but options!
    $self->usage_error("No args allowed") if @$args;
}

sub execute {
    my ($self, $opt, $args) = @_;

    my $result = $opt->{blortex} ? blortex() : blort();

    recheck($result) if $opt->{recheck};

    print $result;
}
```

App::Cmd Drawbacks

- Classic Perl OO
- Not seeing much (any?) development

Other Options

- `Getopt::Std`
- `Getopt::Long::Descriptive`
- `App::Cmd::Simple`
- `MooseX::App::Cmd`
- `(MooseX::)Getopt::Kingpin`
- `MooseX::Getopt::Usage`
- `MooseX::Getopt::Defanged`
- `MooseX::Getopt::Strict`
- `Getopt::EX`
- `Getopt::Alt`

Questions?

Thank you!



Copyright 2019, Jason A. Crome