NAME

App::Prove - Implements the `prove` command.

VERSION

Version 3.17

DESCRIPTION

Test::Harness provides a command, `prove`, which runs a TAP based test suite and prints a report. The `prove` command is a minimal wrapper around an instance of this module.

SYNOPSIS

```
use App::Prove;

my $app = App::Prove->new;
$app->process_args(@ARGV);
$app->run;
```

METHODS

Class Methods

new

Create a new `App::Prove`. Optionally a hash ref of attribute initializers may be passed.

state_class

Getter/setter for the name of the class used for maintaining state. This class should either subclass from `App::Prove::State` or provide an identical interface.

state_manager

Getter/setter for the instance of the `state_class`.

add_rc_file

```
$prove->add_rc_file('myproj/.proverc');
```

Called before `process_args` to prepend the contents of an rc file to the options.

process_args

```
$prove->process_args(@args);
```

Processes the command-line arguments. Attributes will be set appropriately. Any filenames may be found in the `argv` attribute.

Dies on invalid arguments.

run

Perform whatever actions the command line args specified. The `prove` command line tool consists of the following code:

```
use App::Prove;

my $app = App::Prove->new;
$app->process_args(@ARGV);
exit( $app->run ? 0 : 1 );  # if you need the exit code
```

require_harness

Load a harness replacement class.

```
$prove->require_harness($for => $class_name);
```

print_version

Display the version numbers of the loaded *TAP::Harness* and the current Perl.

Attributes

After command line parsing the following attributes reflect the values of the corresponding command line switches. They may be altered before calling `run`.

`archive`
`argv`
`backwards`
`blib`
`color`
`directives`
`dry`
`exec`
`extension`
`failures`
`comments`
`formatter`
`harness`
`ignore_exit`
`includes`
`jobs`
`lib`
`merge`
`modules`
`parse`
`plugins`
`quiet`
`really_quiet`
`recurse`
`rules`
`show_count`
`show_help`
`show_man`
`show_version`
`shuffle`
`state`
`state_class`
`taint_fail`

```
taint_warn
test_args
timer
verbose
warnings_fail
warnings_warn
```

PLUGINS

`App::Prove` provides support for 3rd-party plugins. These are currently loaded at run-time, *after* arguments have been parsed (so you can not change the way arguments are processed, sorry), typically with the `-P`*`plugin`* switch, eg:

```
prove -PMyPlugin
```

This will search for a module named `App::Prove::Plugin::MyPlugin`, or failing that, `MyPlugin`. If the plugin can't be found, `prove` will complain & exit.

You can pass an argument to your plugin by appending an = after the plugin name, eg `-PMyPlugin=foo`. You can pass multiple arguments using commas:

```
prove -PMyPlugin=foo,bar,baz
```

These are passed in to your plugin's `load()` class method (if it has one), along with a reference to the `App::Prove` object that is invoking your plugin:

```
sub load {
    my ($class, $p) = @_;

    my @args = @{ $p->{args} };
    # @args will contain ( 'foo', 'bar', 'baz' )
    $p->{app_prove}->do_something;
    ...
}
```

Note that the user's arguments are also passed to your plugin's `import()` function as a list, eg:

```
sub import {
    my ($class, @args) = @_;
    # @args will contain ( 'foo', 'bar', 'baz' )
    ...
}
```

This is for backwards compatibility, and may be deprecated in the future.

Sample Plugin

Here's a sample plugin, for your reference:

```
package App::Prove::Plugin::Foo;

# Sample plugin, try running with:
# prove -PFoo=bar -r -j3
# prove -PFoo -Q
# prove -PFoo=bar,My::Formatter
```

```
use strict;
use warnings;

sub load {
    my ($class, $p) = @_;
    my @args = @{ $p->{args} };
    my $app  = $p->{app_prove};

    print "loading plugin: $class, args: ", join(', ', @args ), "\n";

    # turn on verbosity
    $app->verbose( 1 );

    # set the formatter?
    $app->formatter( $args[1] ) if @args > 1;

    # print some of App::Prove's state:
    for my $attr (qw( jobs quiet really_quiet recurse verbose )) {
        my $val = $app->$attr;
        $val    = 'undef' unless defined( $val );
        print "$attr: $val\n";
    }

    return 1;
}

1;
```

SEE ALSO

prove, *TAP::Harness*