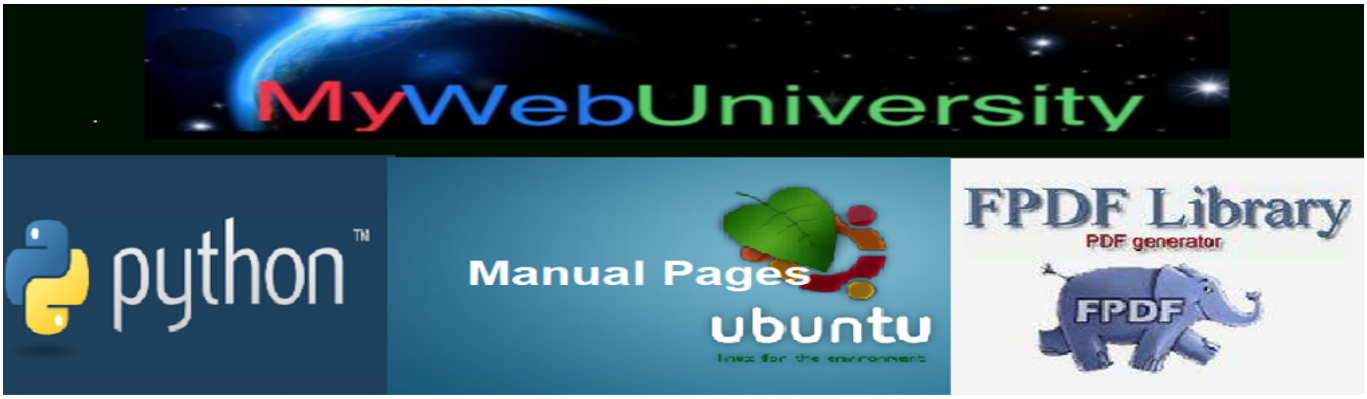

Rocky Enterprise Linux 9.2 Manual Pages on command 'Apache::TestSmoke.3pm'

$ man Apache::TestSmoke.3pm

Apache::TestSmoke(3pm) User Contributed Perl Documentation Apache::TestSmoke(3pm)

NAME

Apache::TestSmoke - Special Tests Sequence Failure Finder

SYNOPSIS

```
# get the usage and the default values
% t/SMOKE -help
# repeat all tests 5 times and save the report into
# the file 'myreport'
% t/SMOKE -times=5 -report=myreport
# run all tests default number of iterations, and repeat tests
# default number of times
% t/SMOKE
# same as above but work only the specified tests
% t/SMOKE foo/bar foo/tar
# run once a sequence of tests in a non-random mode
# e.g. when trying to reduce a known long sequence that fails
% t/SMOKE -order=rotate -times=1 foo/bar foo/tar
# show me each currently running test
# it's not the same as running the tests in the verbose mode
% t/SMOKE -verbose
# run t/TEST, but show any problems after *each* tests is run
# useful for bug reports (it actually runs t/TEST -start, then
# t/TEST -run for each test separately and finally t/TEST -stop
```

```
% t/SMOKE -bug_mode
# now read the created report file
```

DESCRIPTION

The Problem

When we try to test a stateless machine (i.e. all tests are independent), running all tests once ensures that all tested things properly work. However when a state machine is tested (i.e. where a run of one test may influence another test) it's not enough to run all the tests once to know that the tested features actually work. It's quite possible that if the same tests are run in a different order and/or repeated a few times, some tests may fail. This usually happens when some tests don't restore the system under test to its pristine state at the end of the run, which may influence other tests which rely on the fact that they start on pristine state, when in fact it's not true anymore. In fact it's possible that a single test may fail when run twice or three times in a sequence.

The Solution

To reduce the possibility of such dependency errors, it's helpful to run random testing repeated many times with many different srand seeds. Of course if no failures get spotted that doesn't mean that there are no tests inter-dependencies, which may cause a failure in production. But random testing definitely helps to spot many problems and can give better test coverage.

Resolving Sequence Problems

When this kind of testing is used and a failure is detected there are two problems:

1. First is to be able to reproduce the problem so if we think we fixed it, we could verify the fix. This one is easy, just remember the sequence of tests run till the failed test and rerun the same sequence once again after the problem has been fixed.
2. Second is to be able to understand the cause of the problem. If during the random test the failure has happened after running 400 tests, how can we possibly know which previously running tests has caused to the failure of the test 401. Chances are that most of the tests were clean and don't have inter-dependency problem. Therefore it'd be very helpful if we could reduce the long sequence to a minimum. Preferably 1 or 2 tests. That's when we can try to understand the cause of the detected problem.

This utility attempts to solve both problems, and at the end of each iteration print a minimal sequence of tests causing to a failure. This doesn't always succeed, but works in many cases.

This utility:

1. Runs the tests randomly until the first failure is detected. Or non-randomly if the option -order is set to repeat or rotate.
2. Then it tries to reduce that sequence of tests to a minimum, and this sequence still causes to the same failure.
3. (XXX: todo): then it reruns the minimal sequence in the verbose mode and saves the output.
4. It reports all the successful reductions as it goes to STDOUT and report file of the format: smoke-report-<date>.txt.

   In addition the systems build parameters are logged into the report file, so the detected problems could be reproduced.
5. Goto 1 and run again using a new random seed, which potentially should detect different failures.

Reduction Algorithm

Currently for each reduction path, the following reduction algorithms get applied:

1. Binary search: first try the upper half then the lower.
2. Random window: randomize the left item, then the right item and return the items between these two points.

t/SMOKE.PL

t/SMOKE.PL is driving this module, if you don't have it, create it:

```
#!perl
use strict;
use warnings FATAL => 'all';
use FindBin;
use lib "$FindBin::Bin/../Apache-Test/lib";
use lib "$FindBin::Bin/../lib";
use Apache::TestSmoke ();
Apache::TestSmoke->new(@ARGV)->run;
```

usually Makefile.PL converts it into t/SMOKE while adjusting the perl path, but you create t/SMOKE in first place as well.

AUTHOR

Stas Bekman