A Primer on Coverage Analysis with *Devel::Cover*

James E Keenan
(jkeenan@cpan.org)
Perl Seminar New York
Tuesday, December 21, 2004

CPAN Modules and Tests (I)

- Every Perl module on CPAN comes with tests
- Sometimes just a single file
  - *My-Module/test.t*

CPAN Modules and Tests (2)

- More often, a suite of files in their own directory:
  - *My-module/t/01.t*
  - *My-module/t/02.t*
  - *My-module/t/03.t*

So how good are they?

- Good tests would show you where your bugs are.
- If you did revisions to your code, good tests would show you where your revisions broke your code.
- But your tests will never confirm the absence of bugs. Only their presence.

Alternative? Supplement?

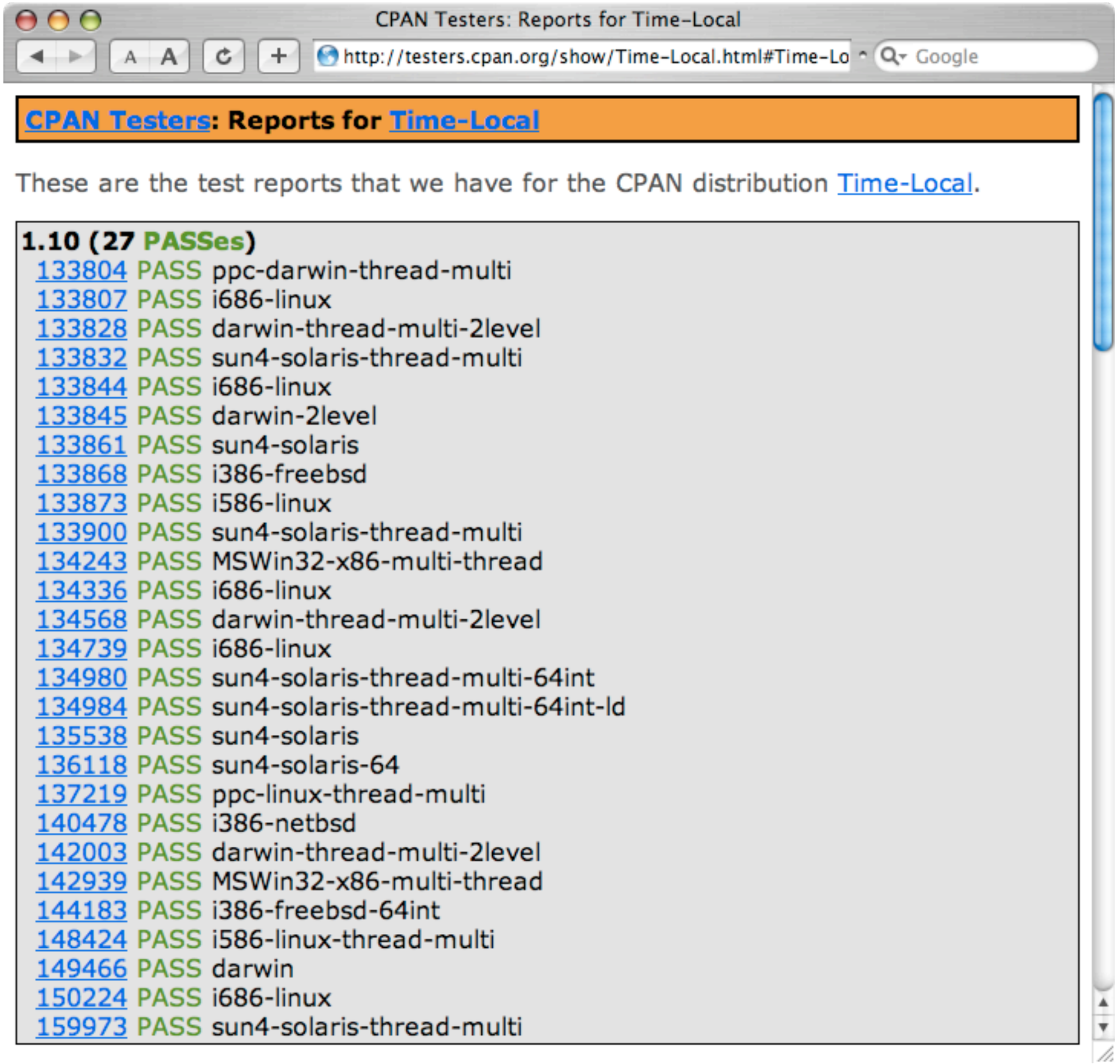
- So you need an alternative ... or, more likely, a supplement.
- Coverage analysis provides that.
- "Code coverage tools measure how thoroughly tests exercise programs."
  -- Brian Marick, "How to Misuse Code Coverage," 1997: www.testing.com

Time::Local

- Objective: "efficiently compute time from local and GMT time."
- It exports only two functions:
  - `timelocal()`
  - `timegm()`
- Inverse of Perl built-in functions
  `localtime()` **and** `gmtime()`

A ‘core’ module

- Must be good: it’s ‘core’
  - Automatically distributed with Perl itself
- Its latest version (1.10) passes all its tests.


CPAN Testers: Reports for Time-Local
http://testers.cpan.org/show/Time-Local.html#Time-Lo
Google
CPAN Testers: Reports for Time-Local
These are the test reports that we have for the CPAN distribution Time-Local.
1.10 (27 PASSes)
133804 PASS ppc-darwin-thread-multi
133807 PASS i686-linux
133828 PASS darwin-thread-multi-2level
133832 PASS sun4-solaris-thread-multi
133844 PASS i686-linux
133845 PASS darwin-2level
133861 PASS sun4-solaris
133868 PASS i386-freebsd
133873 PASS i586-linux
133900 PASS sun4-solaris-thread-multi
134243 PASS MSWin32-x86-multi-thread
134336 PASS i686-linux
134568 PASS darwin-thread-multi-2level
134739 PASS i686-linux
134980 PASS sun4-solaris-thread-multi-64int
134984 PASS sun4-solaris-thread-multi-64int-ld
135538 PASS sun4-solaris
136118 PASS sun4-solaris-64
137219 PASS ppc-linux-thread-multi
140478 PASS i386-netbsd
142003 PASS darwin-thread-multi-2level
142939 PASS MSWin32-x86-multi-thread
144183 PASS i386-freebsd-64int
148424 PASS i586-linux-thread-multi
149466 PASS darwin
150224 PASS i686-linux
159973 PASS sun4-solaris-thread-multi

How good are the tests?

- Just because *Time::Local* passes all its tests doesn't mean the tests are good tests.
- All we can say is: The current tests do not report any bugs.
- How thoroughly do the tests exercise the code?

How much coverage?

- A fair amount of *Time::Local*'s code is not called by its test file ... so it's not tested.
- But first, some more on coverage analysis.

Statement Coverage

- Breaks source code down into individual statements.
- As test suite operates, it records number of times each statement was called.
- Reports which statements were not called in course of test suite
  - -- and, hence, were not tested.

Subroutine Coverage

- Perl modules are packages of subroutines.
- If test suite fails to call a particular subroutine, then that subroutine is, by definition, untested.
- Implication: Making sure each subroutine is tested at least once is a good way to improve coverage of module as a whole.

Branch Coverage

- As program get different inputs, it can take different branches and produce different outputs
- Perl branch points
  - `if ... elsif ... else`
  - `unless ... else`
  - `? :`

Condition Coverage

- Examines situations where truth or falsehood of logical statements are tested.
- Perl conditions:
  - &&
  - ||
- Both branch and condition coverage ask: Did you test all possibilities?

Applying *Devel::Cover* (1)

Easiest approach: Command-line utility: *cover*

```
$ cd Time-Local-1.10
$ perl Makefile.PL
$ make
```

Applying *Devel::Cover* (2)

Delete database created by any previous use of *Devel::Cover* in this directory

```
$ cover -delete
```

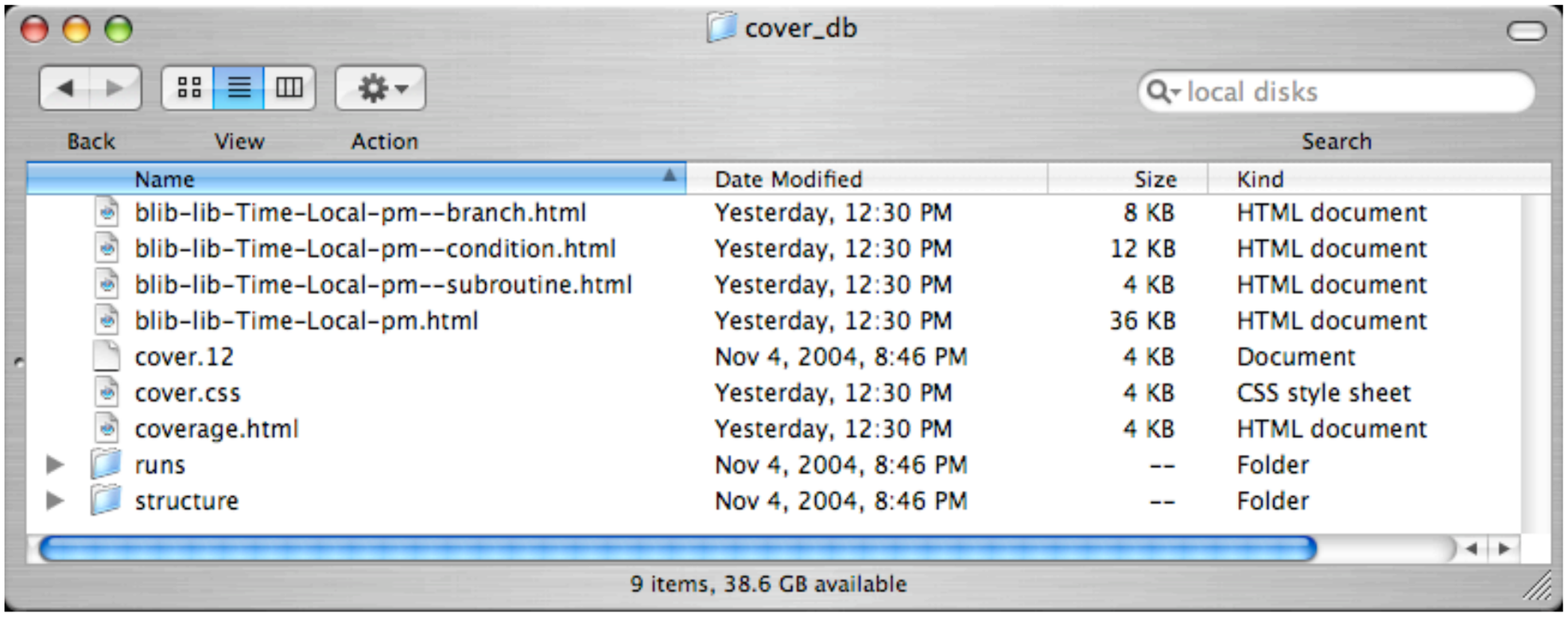
Set special switches and run test suite

```
$ HARNESS_PERL_SWITCHES=-MDevel::
Cover make test
```

Applying *Devel::Cover* (3)

Creates the coverage database and default HTML versions of coverage reports:

```
$ cover
```



Applying *Devel::Cover* (4)

Other *cover* options:

```
$ cover cover_db -report=text

$ cover cover_db -report=text >
My.Module.0.00.coverage.txt

$ cover cover_db -coverage=subroutine

$ cover cover_db -coverage=subroutine
-report=html
```

Making use of coverage analysis (1)

- Print everything out and study it:
  - Code
  - Tests
  - Documentation
- This is “glass-box” testing

Making use of coverage analysis (2)

- Write tests for uncovered subroutines.
- Write tests for uncovered branches and conditions.
- Can usually get 90% statement coverage, often 95% or higher
- 100% statement coverage is usually difficult

A Primer on Coverage Analysis with *Devel::Cover*

James E Keenan
(jkeenan@cpan.org)
Perl Seminar New York
Tuesday, December 21, 2004