

Unit Testing

Doug Treder
UW Extension
Advanced Perl Concepts

Agenda

- Test::Tutorial
- What is a unit test?
- What for?
- Test::Simple
- Test::More
- Test::Class

Test::Tutorial

- AHHHHHHH!!!! NOT TESTING!
- Anything but testing! Beat me, whip me, send me to Detroit, but don't make me write tests!

sob

- Besides, I don't know how to write the darn things.



What's a unit test?

- Programs written to run in batches and test operation of classes.
- Each one sends the class a fixed message and verifies it returns the right answer.
- By “message” I mean each method, plus construction.

What for?

- It's faster
- you won't believe this until it's happened to you; I don't blame you
- You can change everything with much greater confidence

What for?

- This is software engineering
- Since an object is a bundle of capabilities...
- testing its class is writing a test for each capability
- You can write the tests once you've designed the class...
- ...before you write the code!

What for?

- If tests are good, why not test all the time?
- Write tests first, code second.
- Combine tests into a huge suite
- Every time you release, run all the tests (not just your own)
- Whenever you find a bug, add a test for it.
- “test-infected”

Test::Simple

- Here's a basic test:

```
#!/usr/bin/perl  
# baby steps  
  
print 1 + 1 == 2 ? "ok" : "not ok";
```

Test::Simple

- Number it:

```
#!/usr/bin/perl  
# baby steps
```

```
print 1 + 1 == 2 ? "ok 1" : "not ok 1";
```

Test::Simple

- Let perl know how many tests are coming

```
#!/usr/bin/perl  
# baby steps  
  
print "1..1";  
print 1 + 1 == 2 ? "ok 1" : "not ok 1";
```

Test::Simple

- Using Test::Simple :

```
#!/usr/bin/perl

use Test::Simple tests => 1;

ok(1+1 == 2);
```

Test::Simple

- Adding a message:

```
#!/usr/bin/perl

use Test::Simple tests => 1;

ok(1+1 == 2, "testing addition");
```

Organizing Tests

- Place tests in /t
- Subdirectories are ok!
- Add test perl scripts, named with .t
- use “prove” to run the tests

Testing Ooga::Date

- Setup the run first:

```
~/bin/prove
```

```
~/pm/Ooga
```

```
~/pm/Ooga/Date.pm
```

```
~/pm/Ooga/t
```

```
~/pm/Ooga/t/load.t
```

```
% ~/bin/prove -I~/pm ~/pm/Ooga/t
```

```
t/load...FAILED before any test output arrived  
FAILED--1 test script could be run, alas--no  
output ever seen
```

Testing Ooga::Date

- test to see if module loaded:

```
use Test::Simple tests => 1;  
  
use Ooga::Date;  
  
ok(1);
```

Testing Ooga::Date

- test day() function

```
use strict;
use Test::Simple tests => 8;
use Ooga::Date qw(day);
ok(day(0) eq 'ark', 'ark wrong');
ok(day(1) eq 'dip', 'dip wrong');
# ...
ok(day(6) eq 'kir', 'kir wrong');

undef $@;
eval { day(7) };
ok($@, '7 did not die');
```

Test::More

- Test::More is a drop-in replacement for Test::Simple ...with a few new features:
 - `use_ok()`
 - `is()` and `isnt()`
 - `like()` and `unlike()`
 - SKIP:
 - TODO:

Test::More

- use_ok tests loading modules

```
# must be within BEGIN block
```

```
BEGIN { use_ok("MyClass"); }
```

Test::More

- `is()` and `isnt()` give better error messages than `ok()`

```
use Test::More tests => 1;
```

```
is(day(0), 'ark');
```

```
t/day.....#      Failed test (t/day.t at line 8)
```

```
#          got: 'fark'
```

```
#     expected: 'ark'
```

```
# Looks like you failed 1 tests of 6.
```

Test::More

- `is()` and `isnt()` give better error messages than `ok()`
- Error messages still allowed, if you want.

```
use Test::More tests => 1;
```

```
is(day(0), 'ark', "failed the first day!");
```

Test::More

- like() and unlike() use qr() regular expressions

```
use Test::More test => 1;  
use Ooga::Date qw(day);  
  
like(day(5), qr/^[a-z]{3}$/,  
      "day wasn't three letters long");
```

Test::More

- SKIP lets you skip tests successfully for things that are NEVER going to work.

```
use Test::More tests => 2;
SKIP: {
    eval { require HTML::Lint };
    skip "HTML::Lint not installed", 2 if $@;
    my $lint = new HTML::Lint;
    isa_ok( $lint, "HTML::Lint" );
    $lint->parse( $html );
    is( $lint->errors, 0, "No errors found in
HTML" );
}
```

Test::More

- TODO is for writing tests first (a good thing!)
- TODOs that fail are considered successful.

```
TODO: {  
    local $TODO = "haven't implemented long names";  
    is(day(2, "long"), "Wapday");  
}
```

```
not ok 8 # TODO haven't implemented long names  
#     Failed (TODO) test (t/day.t at line 23)  
#           got: 'wap'  
#     expected: 'Wapday'
```

Test::More

```
is ($expression, $value, $description);  
like($attribute, qr/regex/, $description);  
is_deeply($struct1, $struct2, $description);  
isa_ok($object, $class);  
can_ok($object, @methods);  
BEGIN {use_ok($module, @imports) }
```

Assignment

- Create a module ListHelper.pm
 - contains functions sum() and shuffle()
- Write tests first
- Then write the code and run tests til done!

More on CPAN

- Test::Builder
- Test::Exception
- Test::Pod
- Test::Inline
- Test::NoWarnings
- h2xs

Test::Exception

```
#!/usr/bin/perl
use Test::Exception;

lives_ok { risky_function() }
    "risky_function lives!";

dies_ok { $_ / 0 } "div by zero dies!";

throws_ok { some_func() } qr/URL not found/,
    "Nonexistent page get fails";
```

Test::Pod

- Makes sure POD is formatted correctly

```
#!/usr/bin/perl

use strict;
use warnings;
use Test::Pod tests => 1;
use MEBS::MyModule;

pod_ok( $INC{"MEBS/MyModule.pm"} );
```

Test::NoWarnings

- Adds one test - that there are no warnings

```
use Test::More tests => 17;
```

becomes

```
use Test::NoWarnings;  
use Test::More tests => 18;
```

h2xs

- h2xs is a program bundled with perl
- Designed to help write XS extensions
- Sets up the default file structure for CPAN modules
- Adds a default unit test

h2xs

```
dtreder$ h2xs -AXn MEBS::Orc
```

Defaulting to backwards compatibility with perl
5.8.6

If you intend this module to be compatible with
earlier perl versions, please
specify a minimum perl version with the -b option.

```
Writing MEBS-Orc/lib/MEBS/Orc.pm
```

```
Writing MEBS-Orc/Makefile.PL
```

```
Writing MEBS-Orc/README
```

```
Writing MEBS-Orc/t/MEBS-Orc.t
```

```
Writing MEBS-Orc/Changes
```

```
Writing MEBS-Orc/MANIFEST
```

Using the setup

```
dtreder$ cd MEBS-0rc/  
dtreder$ perl Makefile.PL  
Checking if your kit is complete...  
Looks good  
Writing Makefile for MEBS::0rc  
dtreder$ make  
cp lib/MEBS/0rc.pm blib/lib/MEBS/0rc.pm  
Manifying blib/man3/MEBS::0rc.3pm  
dtreder$ make test  
PERL_DL_NONLAZY=1 /usr/bin/perl "-  
MExtUtils::Command::MM" "-e"  
  "test_harness(0, 'blib/lib', 'blib/arch')" t/*.t  
t/MEBS-0rc....ok  
All tests successful.  
Files=1, Tests=1,  0 wallclock secs ( 0.06 cusr +
```

Using prove

- prove is just a program that comes with perl
 - or with the Test::Harness distribution
- prove runs all the tests in a directory
- prints a nice summary
- times them for you too!
- prove is what 'make test' uses when installing many CPAN modules

Using prove

- prove accumulates the 'ok' and 'not ok' results and summarizes them
- Since it's just a perl program you can copy it and use it from anywhere
- There's a copy on the website
- Try it with your ListHelper tests.

Homework

- Use CPAN to install Test modules from class
 - at least, `Test::Exception` and `Test::Pod`
- Quote module
- and its Tests