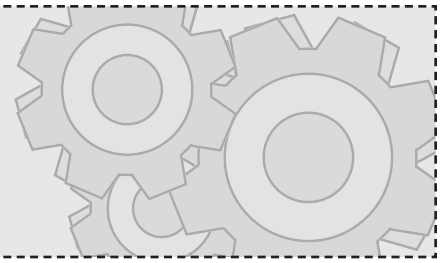
Test Driven Development

Denis Kosykh



[Editor's note: Denis leaves the usual "How to write a test" discussion to the module docs and the tutorials others have already written. Besides the references at the end of the article, you can also use the Perl Testing Reference Card on the back cover of this issue and a short note about Test:: More immediately after this article, on page 20.]

To write a line of code, I have to know what I intend the line to do. Otherwise, the line will not do what I meant. How do I check if code does what it should do? Simply put: I have to express my intention as a test case. Using **Test::Simple**, I can write a test for factorial subroutine:

```
use Test::Simple tests => 1;
ok( factorial(5) == 120 );
```

Test-Driven Development (TDD) specifies code requirements as tests before that code is written. Then I write only the code which gets the tests to pass. That way I have a test suite for all functionality of my code. Even more important that it makes me express my intentions clearly in the tests and then in my code, which results in a robust program.

Fake it

I demonstrate TDD with a simple task: print a positive integer while separating the places with commas. Thus, I have to print the number 40678907654 as 40,678,907,654.

I write tests using **Test::More**, which is a part of the perl distribution starting with version 5.8. I plan for one test.

```
use Test::More tests => 1;
I run it immediately and get a warning:
% perl -w commify.pl
1..1
# No tests run!
```

I specified that I want to execute one test but did not write a test. Thus **Test::More** pointed to the error. I did that on purpose: now I am sure that **Test::More** works for me. With TDD, I run the tests at each step, making sure they fail when they should. Next, I write the simplest test I can think about. This **is()** function from **Test::More** passes if both arguments are the same. In this case, the result of a call to **readable()** and 1.

```
is( readable(1), 1 );
```

It does not compile because readable() does not exist.

Indeed, I used that test to specify that I need a **readable()** subroutine. The test failure motivates new code.

```
sub readable {}
```

Now it compiles, but the test does not pass.

```
1..1
not ok 1
# Failed test (commify at line 2)
# got: undef
# expected: '1'
# Looks like you failed 1 tests of 1.
```

The simplest way to get the test to pass is to return what it wants.

```
sub readable { return 1; }
```

The test passes, but it does not solve the original task. To prove it I write another test with different arguments.

```
is( readable(999), "999" );
```

I don't expect this test to pass, and it doesn't. To make both tests work, **readable()** only has to return its argument.