Day 14: Testing

suggested reading:
Test::Simple, Test::More, Test::Harness documentation
HOMEWORK
Problem

I wrote a script!
\o/
Did I do it right?
>.<
Software Quality

• One model of quality (ISO 9126-1):
  – functionality (incl. security)
  – reliability
  – usability
  – efficiency
  – maintainability
  – portability

• Which is most important?
Testing

• Can check/verify:
  – functionality
  – reliability
  – usability
  – efficiency
  – portability

• Our focus is functionality & reliability
• (May help with portability, too)
Manual Testing

• “Try it out”
• Print/log statements
  – see, e.g., \texttt{Log::Log4perl}
• Debugger
• Code review
• Formal analysis
Automated Testing

• Write software to test other software

• Humans vs. machines

• Types of testing
  – Unit testing
  – Functional testing
  – Performance testing (not covered here)
Test::Simple

• Run a .t file to test a module

```perl
#!/usr/bin/perl
use strict; use warnings;
use SomeModule; # to be tested
use Test::Simple tests => 2;
ok(is_doing_ok(), 'doing ok');
ok(the_result() == 7, 'value ok');
```
Test::Harness

• Create a simple wrapper script

```perl
#!/usr/bin/perl
use strict; use warnings;

use Test::Harness;

my @tests = @ARGV ? @ARGV : <*.t>;
runtests(@tests);
```
use Test::More tests => \textit{nn};

\texttt{ok(is\_ok(), 'is OK');}
\texttt{is($\textit{the\_answer}, 42, 'answer ok');}
\texttt{isnt(exit\_status(), 1, 'syscall');}
\texttt{like($\textit{name}, qr/tim/i, 'good name');}
\texttt{unlike($result, qr/error/i, 'test');}
\texttt{diag('debugging message here');}
\texttt{SKIP: {}
    \texttt{skip 'not available', 1 if \ldots;}
    \texttt{ok(\ldots);}
\texttt{}};}
use Getopt::Long;
GetOptions('test' => \&run_tests);

# main script & subroutines here

sub run_tests {
    require Test::More;
    Test::More-&gt;import;
    plan(tests =&gt; nnn);
    # test subroutines here
    exit 0;
}"
Unit Testing Tips

- Test individual routines in code
- Aim for reasonable coverage
- Run often!
  - After every (significant?) change
  - Certainly before you commit
- Encode prior failures into tests
Functional Testing

• Test complete script as a black box

• Use `system()` and ``

• Check exit codes, output, files, ...

• Other principles apply

• Use to test ANY kind of executable
use Test::More tests => 4;

my $output = `my-script`;
is($?, 0, 'no args exit 0');
like($output, qr/Hello,/, 'no args out');

$output = `my-script --gibberish`;
is($? >> 8, 1, 'bad arg exit 1');
like($output, qr/error/i, 'bad arg out');

...
Test-First Development

• Radical idea: Write tests **FIRST**!

• Then, write code until tests pass

• Can clarify design process

• Becomes persistent record of design

• And of course... is useful for testing!

• Read this:
  http://junit.sourceforge.net/doc/testinfected/testing.htm
Other Languages

• Most based on jUnit
• Expect similar (and richer) assertions
• Introspection rocks!
Ruby Example

```ruby
require 'test/unit'
class TC_MyTest < Test::Unit::TestCase
  def test_this
    assert(blah(), 'blah worked')
    assert_equal(42, answer(), 'quest.')
    assert_match(\d+, input(), 'etc.')
  end
end
```

```ruby
require 'test/unit'
require 'tc_mytest'
```