Testing File Operations in Ruby

Anthony Panozzo
@panozzaj
Indy.rb
20120613

Why might we test file operations?

Why might we test file operations?



Rolling our own

```
def test_creates_directory
  FileUtils.expects(:mkdir).with("directory").once
  Library.add "directory"
end
```

Rolling our own

```
def test_creates_directory
  FileUtils.expects(:mkdir).with("directory").once
  Library.add "directory"
end
```

Issues:

- tightly coupled
- will execute file operations unless always mocked

MockFS gem

http://mockfs.rubyforge.org/

"MockFS is a test-obsessed library for mocking out the entire file system. It provides mock objects that clone the functionality of File, FileUtils, Dir, and other in-Ruby file-access libraries."

MockFS overview

MockFS example

```
def move_log
  MockFS.file_utils.mv('/var/log/httpd/access_log',
                       '/home/francis/logs/')
end
```

MockFS example

```
class TestMoveLog < Test::Unit::TestCase
    def test_move_log
      # Set MockFS to use the mock file system
      MockFS.mock = true

      # Fill certain directories
      MockFS.fill_path '/var/log/httpd/'
      MockFS.fill_path '/home/francis/logs/'

      # Create the access log
      MockFS.file.open( '/var/log/httpd/access_log', File::CREAT ) do |f|
        f.puts "line 1 of the access log"
      end

      # Run the method
      move_log

      # Test that it was moved, along with its contents
      assert( MockFS.file.exist?( '/home/francis/logs/access_log' ) )
      assert( !MockFS.file.exist?( '/var/log/httpd/access_log' ) )
      contents = MockFS.file.open( '/home/francis/logs/access_log' ) do |f|
        f.gets( nil )
      end
      assert_equal( "line 1 of the access log\n", contents )
    end
  end
```

MockFS pros, cons

Pros:

- a way to mock the file system in testing

Cons:

- need to change production code
- introduces testing artifacts
- verbose
- can't shell out and test that

FakeFS gem

https://github.com/defunkt/fakefs

"Mocha is great. But when your library is all about manipulating the filesystem, you really want to test the behavior and not the implementation."

FakeFS Example

```
def test_creates_directory
  FileUtils.expects(:mkdir).with("directory").once
  Library.add "directory"
end
```

versus

```
def test_creates_directory
  Library.add "directory"
  assert File.directory?("directory")
end
```

How do we get this magic?

```
require 'fakefs/safe'
FakeFS.activate!
# your code
FakeFS.deactivate!


FakeFS do
  # your code
end



require 'fakefs/spec_helpers'

describe "my spec" do
  include FakeFS::SpecHelpers
end
```

Magic, continued.

```
module FakeFS
  def self.activate!
    Object.class_eval do
      remove_const(:Dir)
      remove_const(:File)
      remove_const(:FileTest)
      remove_const(:FileUtils)
      remove_const(:Pathname) if RUBY_VERSION >= "1.9.3"


      const_set(:Dir,       FakeFS::Dir)
      const_set(:File,      FakeFS::File)
      const_set(:FileUtils, FakeFS::FileUtils)
      const_set(:FileTest,  FakeFS::FileTest)
      const_set(:Pathname,  FakeFS::Pathname) if RUBY_VERSION >=
"1.9.3"
    end
    true
```

FakeFS Overview

A real-world example

```
context 'when the filepath is an archive file' do
  it 'should extract the archive file to a temp directory if the
archive exists' do
    archive = 'tarfile.tar.gz'
    FileUtils.touch archive
    mock(@installer).install_from_archive archive
    @installer.install_from_file_system archive
  end

  it 'should print an error message and exit if the archive cannot be
found' do
    mock(Railsthemes::Safe).log_and_abort(/Cannot find/)
    @installer.install_from_file_system 'tarfile.tar.gz'
  end
end
```

FakeFS Pros and Cons

Pros:

- Not coupled
- Not leaking testing into production code
- Easy to copy from real FS

Cons:

- Still can't shell and expect it to work
- Slightly buggy

Populating test files

https://github.com/devver/construct

A DSL for creating temporary files and directories during testing

```
class ExampleTest < Test::Unit::TestCase
  include Construct::Helpers

  def test_example
    within_construct do |c|
      c.directory 'alice/rabbithole' do |d|
        d.file 'white_rabbit.txt', "I'm late!"

        assert_equal "I'm late!", File.read('white_rabbit.txt')
      end
    end
  end
end
```

Some other thoughts

Worth it, and quick

Mount file as filesystem

Testing File Operations in Ruby

Anthony Panozzo
@panozzaj
Indy.rb
20120613