# Writing better code using the RCov code coverage tool

**github.com/igal/rcov_tutorial**

Igal Koshevoy, Pragmaticraft
Business-Technology Consultant
igal@pragmaticraft.com
@igalko on Twitter & Identi.ca

## Terminology

- **rcov:** A code coverage tool for Ruby
- **Code coverage report:** Describes what code is executed ("covered") by tests/specs.
- **“X% coverage”:** Percentage of your code covered by tests/specs – more is better.

## Why should you care?

- Code coverage can help you find bugs and missing test cases.
- Useful for:
    1. **Existing code base:** fix and stabilize that bug-ridden mess
    2. **New commits:** catch bugs in the most likely place – new code without coverage

## Caveats

- It’s a tool, not a silver bullet
- 100% code coverage is usually impossible
- Mocks and stubs help cover more code, but prevent execution from reaching the stubbed calls and mocked classes
- Diminishing returns set in somewhere between 0% and 100% code coverage :D
- RCov does line coverage, not path coverage!!! (Discussed later)

## Sample code

Source: github.com/igal/rcov_tutorial

The “mylibrary” directory contains:

- Library that’s being tested/spec’ed
- Spec file that describes library’s behavior
- Rakefile to generate coverage reports for:
  - “**rake -T**” for instructions
  - “**rake rcov**” to create report for all code
  - “**rake rcov:save**” and “**rake rcov:diff**” save state and report new uncovered code

## RCov does line coverage, not path coverage!!!

This succinct line is always shown as covered – it doesn't tell you if 1/0 was run:

```
puts 1/0 unless lucky?
```

Rewriting this as separate lines makes it obvious that 1/0 wasn't run:

```
if lucky?
    puts 1/0
end
```

## I <3 RCov

- Easy and useful way to write better code
- Finds bugs you forgot to deal with
- Finds tests you forgot to write
- Finds problems in code people submitted
- Surprising useful even at high code coverage levels