Unit Testing

I. Questions about testing

A. Why test?
   1. One of the top three ways to become a better developer
   2. It makes it easier and faster to build robust software

B. What is testing?
   1. There are many different types of testing
      a) Unit, functional, and integration testing
      b) Performance and regression testing and quality assurance
      c) Etc.
   2. There are a million ways to do it
      a) Test/Behavior Driven Development
      b) Rspec, minitest, test/unit, and a ton more
      c) Continuous Integration and autotest
   3. For our purposes: some automated testing for our code

C. Testing advantages
   1. Less debugging time and smaller code chunks make it faster
   2. Tests are my memory
      a) Even more important with team development
   3. Environment changes are much easier to handle
   4. It increases my confidence in my software (easier refactoring)

II. A Testable Scenario

A. gnugo --mode gtp --boardsize 9
   1. showboard
   2. play black e4
   3. showboard
   4. genmove white
   5. showboard
   6. play black e4
   7. quit

B. We can wrap this with a library
   1. This is a perfect fit for some unit testing

III. Starting the GTP library

A. Build a project

1. mkdir -p gtp/{lib,test}
2. echo "rvm use 1.9.2" > gtp/.rvmrc
3. cd gtp
4. mate .

B. Build a minimal GTP class

1. initialize() is just IO.popen()
2. quit() is puts(), read(), and close()
3. Stress the need to work on a small scale

C. Add a test

1. Test the return value of the commit method
2. Demonstrate assert(), assert_equal(), and assert_match()
3. Demonstrate failures, errors, and skips

D. Refactor to return success

1. Explain take_while()
2. Explain boolean sub!()
3. Add @id and @error and sub!() response

E. Add and test a method returning a response

1. Write query_boardsize()
2. Test query_boardsize()
3. Simplify tests with gtp() (but show setup() first)
4. Extract send_command()
5. Note the code, test, and refactor cycle

IV. Introduce Test Driven Development

A. Explain Red / Green / Refactor

1. Stress that you don't skip the Red step

B. Test Drive some additional methods

1. play() (and error() as needed) with some error checking
   a) Debug gtp() reuse (use teardown())
2. genmove()
3. showboard()

V. Mocking/Stubbing

A. Our current method of testing this library isn't perfect
   1. Testing against the "database"
      a) Makes some sense in Rails, but not as much here
   2. You have to match our environment to run the tests
   3. We have to be careful to manage resources
   4. It's tricky to manufacture errors
   5. GNU Go is already tested
      a) We really just care about or wrapper of it
   6. It makes it hard to test things like invocation and close()

B. Define mocking and stubbing
   1. Mention some tools for both
      a) minitest/mock, RSpec (built-in), Mocha, etc.

C. Ruby is so dynamic though, they aren't always needed
   1. Build a minimal MockPipe
   2. Reopen and edit GTP
   3. The invoke command and mode
   4. Test Drive adding command-line argument support
   5. Restore the other tests by finishing MockPipe API
      a) Introduce StringIO
      b) Make the IO methods work
      c) Fix responses
      d) Test Drive close() (clean-up teardown())
         (1) Note that this does exercise code in GTP, not just tests