General Tips

These are tips I've given over and over and over and over…

- Use 2 space indent, no tabs.
- Use [] over Array.new.
- Use {} over Hash.new.
- Don't rescue Exception. EVER. or I will stab you.
- Don't call exit inside of a library method.
- Use whitespace more. In both directions.
- Use globals less.
- Use parens to disambiguate, otherwise avoid them.
- Learn to use Enumerable. You will not be a rubyist until you do.
- Returning different types is almost always a no-no.
- dontNameVarsCamelCase, use_readable_variables.
- Hungarian notation is for Other Languages, not Ruby.
- Separate ideas with blank lines, just like paragraphs.
- Align stuff to be cleaner and to optimize for human pattern matching.
- Don't use == to compare floats. Also, learn floats.

See http://github.com/chneukirchen/styleguide/raw/master/RUBY-STYLE for more.

General Syntax Rules

- Comments start with a pound/sharp (#) character and go to EOL.
- Ruby programs are a sequence of expressions.
- Each expression is delimited by semicolons(;) or newlines unless obviously incomplete (e.g. trailing '+').
- Backslashes at the end of line does not terminate expression.

Reserved Words

```
alias   and     BEGIN   begin   break   case    class   def     defined?
do      else    elsif   END     end     ensure  false   for     if
in      module  next    nil     not     or      redo    rescue  retry
return  self    super   then    true    undef   unless  until   when
while   yield
```

Types

Basic types are numbers, strings, ranges, regexen, symbols, arrays, and hashes. Also included are files because they are used so often.

Numbers

```
123
1_234
123.45
1.2e-3
0xffff   # hex
```

```
0b01011  # binary
0377     # octal
?a       # ASCII character (1.8 only -- 1.9 returns a string "a")
?\C-a    # Control-a
?\M-a    # Meta-a
?\M-\C-a # Meta-Control-a
```

Strings

In all of the %( ) cases below, you may use any matching characters or any single character for delimiters. %[], %!!, %@@, etc.

```
'no interpolation'
"#{interpolation}, and backslashes\n"
%q(no interpolation)
%Q(interpolation and backslashes)
%(interpolation and backslashes)
`echo command interpretation with interpolation and backslashes`
%x(echo command interpretation with interpolation and backslashes)
```

Backslashes:

```
\t (tab), \n (newline), \r (carriage return), \f (form feed), \b
(backspace), \a (bell), \e (escape), \s (whitespace), \nnn (octal),
\xnn (hexadecimal), \cx (control x), \C-x (control x), \M-x (meta x),
\M-\C-x (meta control x)
```

Here Docs:

```
<<identifier   - interpolated, goes until identifier
<<"identifier" - same thing
<<'identifier' - no interpolation
<<-identifier  - you can indent the identifier by using "-" in front
```

Encodings:

Waaaay too much to cover here. Try these instead:

- http://nuclearsquid.com/writings/ruby-1-9-encodings/
- http://blog.grayproductions.net/articles/ruby_19s_three_default_encodings

Symbols

Internalized String. Guaranteed to be unique and quickly comparable. Ideal for hash keys.

1.8: Symbols may not contain \0 or be empty.

```
:symbol                        == :symbol
:'#{"without"} interpolation'  == :"#{"without"} interpolation"
:"#{"with"} interpolation"     == :"with interpolation"
%s(#{"without"} interpolation) == :"#{"without"} interpolation"
```

Ranges

```
1..10
1...10
'a'..'z'
```

```
'a'...'z'
(1..10)  === 5   # true
(1..10)  === 10  # true
(1...10) === 10  # false
(1..10)  === 15  # false
```

You can define your own by making them Comparable and implementing `#succ`.

```
class RangeThingy
  include Comparable

  def <=> rhs
    # ...
  end

  def succ
    # ...
  end
end

range = RangeThingy.new(lower_bound)..RangeThingy.new(upper_bound)
```

Regexen

Test out your regexen in irb or on: http://www.rubyxp.com/ or http://rubular.com.

Usual recommended form:

```
str =~ /regex/
```

Lexical options:

```
/normal regex/iomx[neus]
%r{alternate form} # (where {} can be any character XX or pair () [] etc)
```

options:

```
  /i          case insensitive
  /o          only interpolate #{} blocks once
  /m          multiline mode - '.' will match newline
  /x          extended mode - whitespace is ignored
  /[neus]     encoding: none, EUC, UTF-8, SJIS, respectively
```

regex characters:

```
  .             any character except newline
  [ ]           any single character of set
  [^ ]          any single character NOT of set
  *             0 or more previous regular expression
  *?            0 or more previous regular expression (non-greedy)
  +             1 or more previous regular expression
  +?            1 or more previous regular expression (non-greedy)
  ?             0 or 1 previous regular expression
  |             alternation
  ( )           grouping regular expressions
  ^             beginning of a line or string
  $             end of a line or string
  {m,n}         at least m but most n previous regular expression
  {m,n}?        at least m but most n previous regular expression (non-greedy)
```

```
\1-9           nth previous captured group
\&             whole match
\`             pre-match
\'             post-match
\+             highest group matched
\A             beginning of a string
\b             backspace(0x08)(inside[]only)
\b             word boundary(outside[]only)
\B             non-word boundary
\d             digit, same as[0-9]
\D             non-digit
\S             non-whitespace character
\s             whitespace character[ \t\n\r\f]
\W             non-word character
\w             word character[0-9A-Za-z_]
\z             end of a string
\Z             end of a string, or before newline at the end
(?#)           comment
(?:)           grouping without backreferences
(?=)           zero-width positive look-ahead assertion
(?!)           zero-width negative look-ahead assertion
(?>)           nested anchored sub-regexp. stops backtracking.
(?imx-imx)     turns on/off imx options for rest of regexp.
(?imx-imx:re)  turns on/off imx options, localized in group.
(?<=)          zero-width positive look-behind assertion.
(?<!)          zero-width negative look-behind assertion.
```

special character classes:

```
[:alnum:]   alpha-numeric characters
[:alpha:]   alphabetic characters
[:blank:]   whitespace - does not include tabs, carriage returns, etc
[:cntrl:]   control characters
[:digit:]   decimal digits
[:graph:]   graph characters
[:lower:]   lower case characters
[:print:]   printable characters
[:punct:]   punctuation characters
[:space:]   whitespace, including tabs, carriage returns, etc
[:upper:]   upper case characters
[:xdigit:]  hexadecimal digits
```

Arrays

```
[1, 2, 3]
%w(foo bar baz #{1+1}) == ["foo", "bar", "baz", "\#{1+1}"]
%W(foo bar baz #{1+1}) == ["foo", "bar", "baz", "2"]
```

Indexes may be negative, and they index backwards (eg -1 is last element).

Hashes

```
{1=>2, 2=>4, 3=>6}
{ key: val } == { :key => val } # 1.9 only.
```

Files

Common methods include:

```ruby
path = File.join(p1, p2, ... pN) # => "p1/p2/.../pN"

f = File.new("path", "r") # don't use this. Use the block form
File.open("path")      { |f| f.read }
File.open("path", "r") { |f| f.read }
File.open("path", "w") { |f| f.puts "woot" }
File.open("iso-8859-1.txt", "r:iso-8859-1") { |f| ... } # 1.9 open with encoding
File.size("path")  # => 42
File.mtime("path") # => Yesterday

IO.foreach("path") { |line| puts line if line =~ /woot/ }
lines = IO.readlines("path")
```

Mode Strings

`"r"`
: R/O, start of file (default mode).

`"r+"`
: R/W, start of file.

`"w"`
: W/O, truncates or creates.

`"w+"`
: R/W, truncates or creates.

`"a"`
: W/O, end of file or creates.

`"a+"`
: R/W, end of file or creates.

`"b"`
: Binary file mode, in addition to above. DOS/Windows only.

Variables

```ruby
$global_variable
@@class_variable
@instance_variable
CONSTANT
::TOP_LEVEL_CONSTANT
OtherClass::CONSTANT
local_variable
```

Pseudo variables

```ruby
self     # the receiver of the current method
nil      # the sole instance of the Class NilClass (falsey)
true     # the sole instance of the Class TrueClass
false    # the sole instance of the Class FalseClass
__FILE__ # the current source file name.
__LINE__ # the current line number in the source file.
```

Pre-defined variables

Some globals have actual readable names:

```ruby
$DEBUG     # The boolean status of the -d switch.
$FILENAME  # Current input file from ARGF. Same as ARGF.filename.
```

```
$LOAD_PATH # Load path for scripts and binary modules by load or require.
$stderr    # The current standard error output.
$stdin     # The current standard input.
$stdout    # The current standard output.
$VERBOSE   # The verbose flag, which is set by the -v switch.
```

But most don't:

```
$!  # The exception object passed to #raise.
$@  # The stack backtrace generated by the last exception raised.
$&  # Depends on $~. The string matched by the last successful match.
$`  # Depends on $~. The string to the left of the last successful match.
$'  # Depends on $~. The string to the right of the last successful match.
$+  # Depends on $~. The highest group matched by the last successful match.
$1  # Depends on $~. The Nth group of the last successful match. May be > 1.
$~  # The MatchData instance of the last match. Thread and scope local. MAGIC
$=  # The flag for case insensitive. Defaults to nil. Deprecated.
$/  # The input record separator (eg #gets). Defaults to newline.
$\  # The output record separator (eg #print and IO#write). Default is nil.
$,  # The output field separator for the print and Array#join. Defaults to nil.
$;  # The default separator for String#split. See -F flag.
$.  # The current line number of the last file from input.
$<  # See ARGF.
$>  # The default output for print, printf. Defaults to $stdout.
$_  # The last input line of string by gets or readline. Thread and scope local.
$0  # Contains the name of the script being executed. May be assignable.
$*  # See ARGV.
$$  # The process number of the Ruby running this script. Read only.
$?  # The status of the last executed child process. Read only. Thread local.
$:  # See $LOAD_PATH.
$"  # The array contains the module names loaded by require.
```

Many command line arguments have an associated global, which is usually just an alias to a real global:

```
$-0  # See $/.
$-a  # Autosplit mode. True if option -a is set. Read-only variable.
$-d  # See $DEBUG.
$-F  # See $;.
$-i  # In in-place-edit mode, this variable holds the extension, otherwise nil.
$-I  # See $LOAD_PATH.
$-l  # True if option -l is set. Read-only.
$-p  # True if option -p is set. Read-only.
$-v  # See $VERBOSE.
$-w  # True if option -w is set.
```

require "English"

```
# Small Medium Large

  $!            $ERROR_INFO
  $@            $ERROR_POSITION
  $;    $FS     $FIELD_SEPARATOR
  $,    $OFS    $OUTPUT_FIELD_SEPARATOR
  $/    $RS     $INPUT_RECORD_SEPARATOR
  $\    $ORS    $OUTPUT_RECORD_SEPARATOR
  $.    $NR     $INPUT_LINE_NUMBER
  $_            $LAST_READ_LINE
  $>            $DEFAULT_OUTPUT
  $<            $DEFAULT_INPUT
  $$    $PID    $PROCESS_ID
```

```
$?            $CHILD_STATUS
$~            $LAST_MATCH_INFO
$=            $IGNORECASE
$*            $ARGV
$&            $MATCH
$`            $PREMATCH
$'            $POSTMATCH
$+            $LAST_PAREN_MATCH
```

Pre-defined global constants

```
STDIN             # The standard input. The default value for $stdin.
STDOUT            # The standard output. The default value for $stdout.
STDERR            # The standard error output. The default value for $stderr.
ENV               # The hash contains current environment variables. Writable.
ARGF              # A meta-IO across all files in ARGV. (eg ARGF.each_line...)
ARGV              # An array of all the arguments given on run.
DATA              # An IO pointing just after __END__ of the running script.
RUBY_ENGINE       # The ruby implementation you're running (eg ruby, rubinius, etc)
RUBY_PLATFORM     # The platform identifier.
RUBY_VERSION      # The ruby version string (VERSION was deprecated).
```

Expressions

Operators and Precedence

```
(Top to bottom)
:: .
[]
**
-(unary) +(unary) ! ~
*  /  %
+  -
<<  >>
&
|  ^
>  >=  <  <=
<=> == === != =~ !~
&&
||
.. ...
=(+=, -=...)
not
and or
```

TODO: add = and others

All of the above are just methods except these:

```
=, ::, ., .., ..., !, not, &&, and, ||, or, !=, !~
```

In addition, assignment operators(+= etc.) are not user-definable.

NOTE: 1.9 has a *horrible* extension to allow you to define !=, !~, !, and not. A special place in hell is reserved for you if you define any of these.

Control Expressions

```
if bool-expr [then]
  body
elsif bool-expr [then]
  body
else
  body
end

unless bool-expr [then]
  body
else
  body
end

expr if     bool-expr
expr unless bool-expr

bool-expr ? true-expr : false-expr

case target-expr
when comparison [, comparison]... [then]
  body
when comparison [, comparison]... [then]
  body
# ...
else # optional else
  body
end
```

Case comparisons may be regexen, classes, whatever. Uses #===.

```
loop do
  body
end

while bool-expr [do]
  body
end

until bool-expr [do]
  body
end

begin
  body
end while bool-expr

begin
  body
end until bool-expr

for name [, name]... in expr [do]
  body
end

expr.each do | name [, name]... | # preferred form over `for`
  body
end
```

```
expr while bool-expr
expr until bool-expr

break # terminates loop immediately.
redo  # immediately repeats w/o rerunning the condition.
next  # starts the next iteration through the loop.
retry # restarts the loop, rerunning the condition.
```

Invoking a Method

Nearly everything available in a method invocation is optional, consequently the syntax is very difficult to follow. Here are some examples:

```
method
obj.method
Class::method # don't use this
Class.method

method(key1 => val1, key2 => val2) # one hash arg, not 2

method(arg1, *[arg2, arg3]) == method(arg1, arg2, arg3)

# As ugly as you want it to be:
method(arg1, key1 => val1, key2 => val2, *splat_arg) #{ block }
```

The argument syntax is fairly complex:

```
  invocation := [receiver ('::' | '.')] name [ parameters ] [ block ]
  parameters := ( [param]* ['=' hashlist] ['*' array] [&aProc] )
  block      := '{' blockbody '}' | 'do' blockbody 'end'
```

Defining a Class

Class names begin w/ capital character.

```
class Identifier [< superclass ]
  expr..
end

# singleton classes, add methods to a single instance
class << obj
  expr..
end
```

Defining a Module

```
module Identifier
  expr..
end
```

Defining a Method

```
def method_name(arg_list, *list_expr, &block_expr)
  expr..
end
```

```ruby
# singleton method
def expr.identifier(arg_list, *list_expr, &block_expr)
  expr..
end
```

- All items of the arg list, including parens, are optional.
- Arguments may have default values (name=expr).
- Method_name may be operators (see above).
- The method definitions can not be nested.
- Methods may override operators: `|`, `^`, `&`, `<=>`, `==`, `===`, `=~`, `>`, `>=`, `<`, `<=`, `+`, `-`, `*`, `/`, `%`, `**`, `<<`, `>>`, `~`, `+@`, `-@`, `[]`, `[]=` (2 args)

Access Restriction

`public`
: totally accessible.

`protected`
: accessible only by instances of class and direct descendants. Even through hasA relationships. (see below)

`private`
: accessible only by instances of class (must be called nekkid no “self.” or anything else).

```ruby
class A
  # Restriction used w/o arguments set the default access control.
  protected

  def protected_method
    # nothing
  end
end

class B < A
  def test_protected
    myA = A.new
    myA.protected_method
  end

  # Used with arguments, sets the access of the named methods and constants.
  public :test_protected
end

b = B.new.test_protected
```

Accessors

Class Module provides the following utility methods:

```ruby
attr_reader   :attribute [, :attribute]...  # Creates reader methods
attr_writer   :attribute [, :attribute]...  # Creates setter methods
attr_accessor :attribute [, :attribute]...  # Creates both readers and writers
```

Aliasing

```ruby
alias          new   old # symbol syntax not needed... bewilderingly
alias         :new  :old # comma not needed either... go figure
alias_method :new, :old
```

Creates a new reference to whatever old referred to. old can be any existing method, operator, global. It may not be a local, instance, constant, or class variable.

Blocks, Closures, and Procs

Blocks/Closures

- blocks must follow a method invocation:
  ```
  invocation do ... end
  invocation { ... }
  ```
- Blocks remember their variable context, and are full closures.
- Blocks are invoked via yield and may be passed arguments.
- Brace form has higher precedence and will bind to the last parameter if invocation made w/o parens.
- do/end form has lower precedence and will bind to the invocation even without parens.

Proc Objects

Created via:

```
proc      { |args| ... } # {} or do/end
Proc.new  { |args| ... }
lambda    { |args| ... }
-> (args) { ... }        # 1.9+ only
&:method_name            # calls Symbol#to_proc creating: proc { |o| o.method_name }

# or by invoking a method w/ a block argument and catching it on the
# calling side with a &block_arg:

def my_method &block
  block.call 42
end

obj.my_method { |o| ... }

# in 1.9+, Proc aliases #=== to #call so you can use them as case conditions:

case []
when :empty?.to_proc then
  # ...
when -> (o) { o > 42 && o.prime? } then
  # ...
end
```

See class Proc for more information.

Exceptions, Catch, and Throw

- Exception
  - NoMemoryError
  - ScriptError
    - LoadError
    - NotImplementedError
    - SyntaxError
  - SecurityError (1.9: move!)
  - SignalException
    - Interrupt

- StandardError [default for plain rescue]
  - ArgumentError
  - EncodingError (1.9)
    - Encoding::CompatibilityError (1.9)
    - Encoding::ConverterNotFoundError (1.9)
    - Encoding::InvalidByteSequenceError (1.9)
    - Encoding::UndefinedConversionError (1.9)
  - FiberError (1.9)
  - IOError
    - EOFError
  - IndexError
    - KeyError (1.9)
    - StopIteration
  - LocalJumpError
  - Math::DomainError (1.9)
  - NameError
    - NoMethodError
  - RangeError
    - FloatDomainError
  - RegexpError
  - RuntimeError [default for plain raise]
  - SecurityError (1.8: move!)
  - SystemCallError
    - Errno::*
  - SystemStackError (1.8: move!)
  - ThreadError
  - TypeError
  - ZeroDivisionError
- SystemExit
- SystemStackError (1.9: move!)
- fatal

Raising and Rescuing

```
raise ExceptionClass[, "message"]
```

```
begin
  expr...
[rescue [error_type [=> var],...]
  expr...] ...
[else
  expr...]
[ensure
  expr...]
end
```

Catch and Throw

```
catch :label do
  do_stuff
  throw :label if condition?
  do_other_stuff
end
```

- `throw :label` jumps back to matching catch and terminates the block.
  - can be external to catch, but has to be reached via calling scope.
  - Hardly ever needed.

Standard Library

Ruby comes with an extensive library of classes and modules. Some are built-in, and some are part of the standard library. You can distinguish the two by the fact that the built-in classes are in fact, built-in. There are no dot-rb files for them.

This list is not comprehensive. Use `ri Class_and/or_method` to look up documentation or try http://ruby-doc.org.

Built-in Library

Class Hierarchy

- Object
  - ARGF.class (1.9)
  - Array
  - Binding
  - 1.8: Continuation
  - Data
    - Encoding::Converter (1.9)
    - NameError::message
  - Dir
  - Encoding (1.9)
  - Enumerator (1.9)
  - 1.8: Enumerable::Enumerator
  - Enumerator::Generator (1.9)
  - Enumerator::Yielder (1.9)
  - Exception (see above for full tree)
  - FalseClass
  - Fiber (1.9)
  - File::Stat
  - Hash
  - IO
    - File
  - MatchData
  - Method
  - Module
    - Class
  - Mutex (1.9: built-in, 1.8: `require "thread"`)
  - NilClass
  - Numeric
    - Complex (1.9: built-in, 1.8: `require "complex"`)
    - Float
    - Integer

        - Bignum
        - Fixnum
      - Rational (1.9: built-in, 1.8: `require "rational"`)
    - Proc
    - Process::Status
    - Random (1.9)
    - Range
    - Regexp
    - RubyVM (1.9)
    - RubyVM::Env (1.9)
    - RubyVM::InstructionSequence (1.9)
    - String
    - Struct
      - Struct::Tms
    - Symbol
    - Thread
    - ThreadGroup
    - Time
    - TrueClass
    - UnboundMethod

Modules

- Comparable
- Enumerable
- Errno
- FileTest
- GC
- Kernel
- Marshal
- Math
- ObjectSpace
- Precision
- Process

Standard Library

The essentials:

benchmark.rb
: a simple benchmarking utility

cgi-lib.rb
: CGI data - simpler than cgi.rb

cgi.rb
: CGI interaction

date.rb
: date object (compatible)

debug.rb
: ruby debugger

delegate.rb
: delegate messages to other object

English.rb
: access global variables by english names

fileutils.rb
: file utility methods for copying, moving, removing, etc.

find.rb
: traverse directory tree

jcode.rb
: UTF-8 and Japanese String helpers (replaces String methods)

net/*
: Networking classes of all kinds

observer.rb
: observer design pattern library (provides Observable)

open-uri.rb
: good wrapper for net/http, net/https and net/ftp

open3.rb
: open subprocess connection stdin/stdout/stderr

ostruct.rb
: python style object (freeform assignment to instance vars)

parsearg.rb
: argument parser using getopts

pp
: prettier debugging output, ‘p’ on steroids.

profile.rb
: ruby profiler - find that slow code!

pstore.rb
: persistent object strage using marshal

singleton.rb
: singleton design pattern library

stringio
: lets you use an IO attached to a string.

tempfile.rb
: temporary file that automatically removed

minitest/*
: 1.9: unit testing framework. (see below)

test/unit
: unit testing framework (1.9: compat library built on top of minitest)

time.rb
: extension to Time class with a lot of converters

tracer.rb
: execution tracer

webrick
: Fairly spiffy web server

yaml
: alternative readable serialization format

Minitest

Minitest ships with 1.9 by default. You can ensure you have the latest code by installing the latest `minitest` gem. Minitest will automatically look for the latest minitest gem if you have any installed.

Unit Test Example

```
require "minitest/autorun"
require "noun"

class TestNoun < MiniTest::Unit::TestCase
  def setup
    @noun = Noun.new
  end

  def test_verb
    assert_equal 42, @noun.verb
  end

  # ... more tests ...
end
```

Unit Spec Example

```
require "minitest/autorun"
require "noun"

describe Noun do
  before do
    @noun = Noun.new
  end

  it "verbs the noun" do
    @noun.verb.must_equal 42
  end

  describe "is nestable noun" do
    # ...
  end
end
```

Assertions

Every assertion (except assert_silent) takes an optional message argument at the end. But they also build their own messages on failure, so you really don't need to provide one except to disambiguate things.

```
assert truthiness

assert_equal :expected_value, object.result
assert_same expected, object.result
assert_nil object.result
assert_in_delta 42.0, object.number
assert_in_epsilon 42.0, object.number

assert_match(/matcher/, any_obj_not_just_strings) # uses =~
assert_empty collection_or_string
assert_includes object.collection_or_string, :expected_element

assert_instance_of Array, collection
```

```
assert_kind_of Enumerable, collection
assert_respond_to object, :method

assert_operator object.result, :truthy?
assert_operator object.result, :<=, 42
assert_predicate object.result, :truthy?
assert_send [recv, msg, arg1, arg2]

assert_output("did something", "") { object.do_something_talky }
assert_silent { object.do_something_quiet }

assert_raises(MyException) { object.do_something_bad }
assert_throws(:my_throw) { object.do_something_throwy }
```

Refutations

Not every assertion has a corresponding refutation. Some simply don't make sense (eg refute_raises – any unexpected exception is automatically an error) or don't lend any value because they don't actually validate behavior / side effects (refute_silent – great… it output *something*… but what?).

```
refute falsiness

refute_equal :unexpected_value, object.result
refute_same expected, object.result
refute_nil object.result
refute_in_delta 42.0, object.number
refute_in_epsilon 42.0, object.number

refute_match(/matcher/, any_obj_not_just_strings) # still uses =~
refute_empty collection_or_string
refute_includes collection, :unexpected_element

refute_instance_of Array, not_a_collection
refute_kind_of Enumerable, not_a_collection
refute_respond_to object, :method

refute_operator object.result, :falsey?
refute_operator object.result, :<=, 42
refute_predicate object.result, :falsey?
```

Expectations

All expectations (positive or otherwise) map to their corresponding assertion/refutation above.

```
object.result.must_equal 42
object.result.must_be_same_as expected_object
object.result.must_be_nil
object.number.must_be_close_to 42.0
object.number.must_be_within_epsilon 42.0

object.collection.must_be_empty
object.collection.must_include :expected_element
object.any_obj_not_just_strings.must_match matcher

object.result.must_be_instance_of Array
object.collection.must_be_kind_of Enumerable
object.must_respond_to :message

object.result.must_be :<=, 42
object.collection_or_string.must_be :empty?
```

```
proc { object.do_something_bad }.must_raise exception
proc { object.do_something_throwy }.must_throw :my_throw

proc { object.do_something_talky }.must_output "something"
proc { object.do_something_quiet }.must_be_silent
```

Negative Expectations

```
object.result.wont_equal 42
object.result.wont_be_same_as unexpected_object
object.result.wont_be_nil
object.number.wont_be_close_to 42.0
object.number.wont_be_within_epsilon 42.0

object.collection.wont_be_empty
collection.wont_include :unexpected_element
object.any_obj_not_just_strings.wont_match matcher

object.result.wont_be_instance_of Array
object.not_a_collection.wont_be_kind_of Enumerable

object.result.wont_be :<=, 42
object.collection_or_string.wont_be :empty?
object.wont_respond_to :message
```

Helper methods

Usable in tests or specs.

```
out, err = capture_io { object.do_something_talky }
out, err = capture_subprocess_io { cmd arg }
flunk "This totally fails"
pass "OCD people need assertion counts to rise"
```

Tools

ruby

Command Line Options

```
  -0[octal]       specify record separator (\0, if no argument).
  -a              autosplit mode with -n or -p (splits $_ into $F).
  -c              check syntax only.
  -Cdirectory     cd to directory, before executing your script.
  --copyright     print the copyright and exit.
  -d              set debugging flags (set $DEBUG to true).
  -e 'command'    one line of script. Several -e's allowed.
  -F regexp       split() pattern for autosplit (-a).
  -h                prints summary of the options.
  -i[extension]   edit ARGV files in place (make backup if extension supplied).
  -Idirectory     specify $LOAD_PATH directory (may be used more than once).
  -Kkcode         specifies KANJI (Japanese) code-set.
```

```
-l              enable line ending processing.
-n              assume 'while gets(); ... end' loop around your script.
-p              assume loop like -n but print line also like sed.
-rlibrary       require the library, before executing your script.
-s              enable some switch parsing for switches after script name.
-S              look for the script using PATH environment variable.
-T[level]       turn on tainting checks.
-v              print version number, then turn on verbose mode.
--version       print the version and exit.
-w              turn warnings on for your script.
-x[directory]   strip off text before #! line and perhaps cd to directory.
-X directory    causes Ruby to switch to the directory.
-y              turns on compiler debug mode.
```

Environment Variables

```
DLN_LIBRARY_PATH Search path for dynamically loaded modules.
RUBYLIB          Additional search paths.
RUBYLIB_PREFIX   Add this prefix to each item in RUBYLIB. Windows only.
RUBYOPT          Additional command line options.
RUBYPATH         With -S, searches PATH, or this value for ruby programs.
RUBYSHELL        Shell to use when spawning. (Windows (and OS/2!) only)
```

irb

```
irb [options] [script [args]]
```

The essential options are:

```
-d              Sets $DEBUG to true. Same as "ruby -d ..."
-f              Prevents the loading of ~/.irb.rc.
-h              Get a full list of options.
-m              Math mode. Overrides --inspect. Loads "mathn.rb".
-r module       Loads a module. Same as "ruby -r module ..."
-v              Prints the version and exits.
--inf-ruby-mode Turns on emacs support and turns off readline.
--inspect       Turns on inspect mode. Default.
--noinspect     Turns off inspect mode.
--noprompt      Turns off the prompt.
--noreadline    Turns off readline support.
--prompt        Sets to one of 'default', 'xmp', 'simple', or 'inf-ruby'.
--readline      Turns on readline support. Default.
--tracer        Turns on trace mode.
```

Besides arbitrary ruby commands, the special commands are:

```
exit                  exits the current session, or the program
fork block            forks and runs the given block
cb args               changes to a secified binding
source file           loads a ruby file into the session
irb [obj]             starts a new session, with obj as self, if specified
conf[.key[= val]]     access the configuration of the session
jobs                  lists the known sessions
fg session            switches to the specifed session
kill session          kills a specified session
```

Session may be specified via session#, thread-id, obj, or self.

Debugger

To invoke the debugger:

```
ruby -r debug ...
```

To use the debugger:

```
b[reak] [file:|class:]<line|method
b[reak] [class.]<line|method
                         set breakpoint to some position
wat[ch] expression       set watchpoint to some expression
cat[ch] exception        set catchpoint to an exception
b[reak]                  list breakpoints
cat[ch]                  show catchpoint
del[ete][ nnn]           delete some or all breakpoints
disp[lay] expression     add expression into display expression list
undisp[lay][ nnn]        delete one particular or all display expressions
c[ont]                   run until program ends or hit breakpoint
s[tep][ nnn]             step (into methods) one line or till line nnn
n[ext][ nnn]             go over one line or till line nnn
w[here]                  display frames
f[rame]                  alias for where
l[ist][ (-|nn-mm)]       list program, - lists backwards
                         nn-mm lists given lines
up[ nn]                  move to higher frame
down[ nn]                move to lower frame
fin[ish]                 return to outer frame
tr[ace] (on|off)         set trace mode of current thread
tr[ace] (on|off) all     set trace mode of all threads
q[uit]                   exit from debugger
v[ar] g[lobal]           show global variables
v[ar] l[ocal]            show local variables
v[ar] i[nstance] object  show instance variables of object
v[ar] c[onst] object     show constants of object
m[ethod] i[nstance] obj  show methods of object
m[ethod] class|module    show instance methods of class or module
th[read] l[ist]          list all threads
th[read] c[ur[rent]]     show current thread
th[read] [sw[itch]] nnn  switch thread context to nnn
th[read] stop nnn        stop thread nnn
th[read] resume nnn      resume thread nnn
p expression             evaluate expression and print its value
h[elp]                   print this help
everything else          evaluate
empty                    repeats the last command
```