

Ruby QuickRef

Contents:

- [Contents](#)
- [Language](#)
 - [General Syntax Rules](#)
 - [Reserved words](#)
 - [Types](#)
 - [Numbers](#)
 - [Strings](#)
 - [Backslashes](#)
 - [Here Docs](#)
 - [Ranges](#)
 - [Regexen](#)
 - [Arrays](#)
 - [Hashes](#)
 - [Files](#)
 - [Mode Strings](#)
 - [Variables](#)
 - [Pseudo variables](#)
 - [Pre-defined variables](#)
 - [Pre-defined global constants](#)
 - [Expressions](#)
 - [Terms](#)
 - [Operators and Precedence](#)
 - [Control Expressions](#)
 - [Invoking a Method](#)
 - [Defining a Class](#)
 - [Defining a Module](#)
 - [Defining a Method](#)
 - [Access Restriction](#)
 - [Accessors](#)
 - [Aliasing](#)
 - [Blocks, Closures, and Procs](#)
 - [Blocks/Closures](#)
 - [Proc Objects](#)
 - [Exceptions, Catch, and Throw](#)
- [Standard Library](#)
 - [Built-in Library](#)
 - [Class Hierarchy](#)
 - [Modules](#)
 - [Standard Library](#)
- [Tools](#)
 - [ruby](#)
 - [Command Line Options](#)
 - [Environment Variables](#)
 - [irb](#)
 - [xmp](#)
 - [ruby-mode](#)
 - [Debugger](#)
 - [rdoc](#)
- [Mindshare, Idiom and Patterns](#)
 - [Object Design](#)
 - [Visitor Pattern](#)
 - [Class SimpleDelegator, DelegateClass](#)
 - [Module Observer](#)

- [Module Singleton](#)
- [Other Third-party Libraries](#)
 - [Racc](#)
 - [Test::Unit](#)

Language

General Syntax Rules

- Comments start with a pound/sharp (#) character and go to EOL.
- Ruby programs are 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
?\C-a   Control-a
?\M-a   Meta-a
?\M-\C-a Meta-Control-a
:symbol Integer corresponding to identifiers, variables, and operators.
```

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
```

Ranges

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

```

while gets # prints lines starting at 'start' and ending at 'end'
  print if /start/../end/
end

class RangeThingy
  # ...
  def <=>(rhs)
    # ...
  end
  def succ
    # ...
  end
end
range = RangeThingy.new(lower_bound)..RangeThingy.new(upper_bound)

```

Regexen

```

/normal regex/iomx[neus]
%r|alternate form|

```

options:

```

/i          case insensitive
/o          case insensitive
/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)
\A        beginning of a string
\b        backspace(0x08)(inside[ ]only)
\B        non-word boundary
\b        word boundary(outside[ ]only)
\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
(?ix-ix)  turns on/off i/x options, localized in group if any.
(?ix-ix: ) turns on/off i/x options, localized in non-capturing group.

```

Arrays

```

[1, 2, 3]
%w(foo bar baz)
%W(foo bar baz #{var})

```

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

Hashes

```
{1=>2, 2=>4, 3=>6}
{ expr => expr...}
```

Files

Common methods include:

- File.join(p1, p2, ... pN) => "p1/p2/.../pN" platform independent paths
- File.new(path, modestring="r") => file
- File.new(path, modenum [, permnum]) => file
- File.open(fileName, aModeString="r") {file| block} -> nil
- File.open(fileName [, aModeNum [, aPermNum]]) {file| block} -> nil
- IO.foreach(path, sepstring=\$/) {line| block}
- IO.readlines(path) => array

Mode Strings

```
r
  Read-only, starts at beginning of file (default mode).
r+
  Read-write, starts at beginning of file.
w
  Write-only, truncates existing file to zero length or creates a new file for writing.
w+
  Read-write, truncates existing file to zero length or creates a new file for reading and writing.
a
  Write-only, starts at end of file if file exists, otherwise creates a new file for writing.
a+
  Read-write, starts at end of file if file exists, otherwise creates a new file for reading and writing.
b
  (DOS/Windows only) Binary file mode (may appear with any of the key letters listed above).
```

Variables

```
$global_variable
@instance_variable
[OtherClass:]CONSTANT
local_variable
```

Pseudo variables

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

Pre-defined variables

```
$_        The exception information message set by 'raise'.
$@        Array of backtrace of the last exception thrown.
$&        The string matched by the last successful pattern match in this scope.
$`        The string to the left of the last successful match.
$'        The string to the right of the last successful match.
$+        The last bracket matched by the last successful match.
$1        The Nth group of the last successful match. May be > 1.
$~        The information about the last match in the current scope.
$=        The flag for case insensitive, nil by default.
$/        The input record separator, newline by default.
$\        The output record separator for the print and IO#write. Default is nil.
$,        The output field separator for the print and Array#join.
$;        The default separator for String#split.
$.        The current input line number of the last file that was read.
$<        The virtual concatenation file of the files given on command line.
```

```

$>          The default output for print, printf. $stdout by default.
$_          The last input line of string by gets or readline.
$0          Contains the name of the script being executed. May be assignable.
$*          Command line arguments given for the script sans args.
$$          The process number of the Ruby running this script.
$?          The status of the last executed child process.
$:          Load path for scripts and binary modules by load or require.
$"          The array contains the module names loaded by require.
$DEBUG      The status of the -d switch.
$FILENAME   Current input file from $<. Same as $<.filename.
$LOAD_PATH The alias to the $:.
$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.
$-0         The alias to $/.
$-a         True if option -a is set. Read-only variable.
$-d         The alias to $DEBUG.
$-F         The alias to $;.
$-i         In in-place-edit mode, this variable holds the extension, otherwise nil.
$-I         The alias to $:.
$-l         True if option -l is set. Read-only variable.
$-p         True if option -p is set. Read-only variable.
$-v         The alias to $VERBOSE.

```

Pre-defined global constants

```

TRUE        The typical true value.
FALSE       The false itself.
NIL         The nil itself.
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.
ARGV        The alias to the $<.
ARGV        The alias to the $*.
DATA        The file object of the script, pointing just after __END__.
RUBY_VERSION The ruby version string (VERSION was deprecated).
RUBY_RELEASE_DATE The release date string.
RUBY_PLATFORM The platform identifier.

```

Expressions

Terms

Terms are expressions that may be a basic type (listed above), a shell command, variable reference, constant reference, or method invocation.

Operators and Precedence

```

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

```

All of the above are just methods except these:

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

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

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

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

```

(comparisons may be regexen)

```

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]... |
  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
- method(arg1, arg2)
- method(arg1, key1 => val1, key2 => val2, aval1, aval2) #{ block }
- method(arg1, *[arg2, arg3]) becomes: method(arg1, arg2, arg3)

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

Defining a Class

Classnames 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

# 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.
- Restriction used w/o arguments set the default access control.
- Used with arguments, sets the access of the named methods and constants.

```
class A
  protected
  def protected_method
    # nothing
  end
end
class B < A
  public
  def test_protected
    myA = A.new
  end
end
```

```

    myA.protected_method
  end
end
b = B.new.test_protected

```

Accessors

Class Module provides the following utility methods:

`attr_reader <attribute>[, <attribute>]...`

Creates a read-only accessor for each <attribute>.

`attr_writer <attribute>[, <attribute>]...`

Creates a write-only accessor for each <attribute>.

`attr <attribute> [, <writable>]`

Equivalent to "attr_reader <attribute>; attr_writer <attribute> if <writable>"

`attr_accessor <attribute>[, <attribute>]...`

Equivalent to "attr <attribute>, TRUE" for each argument.

Aliasing

```

alias      :old :new
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:

- Kernel#proc
- Proc#new
- By invoking a method w/ a block argument.

See class Proc for more information.

Exceptions, Catch, and Throw

- Exception
 - StandardError
 - LocalJumpError
 - SystemStackError
 - ZeroDivisionError
 - RangeError
 - FloatDomainError
 - SecurityError
 - ThreadError
 - IOError
 - EOFError
 - ArgumentError
 - IndexError

- RuntimeError
- TypeError
- SystemCallError
 - Errno::*
- RegexpError
- SignalException
- Interrupt
- fatal
- NoMemoryError
- ScriptError
 - LoadError
 - NameError
 - SyntaxError
 - NotImplementedError
- SystemExit

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

The default error_type for rescue is StandardError, not Exception.

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.

Built-in Library

Class Hierarchy

- Object
 - Hash
 - Symbol
 - IO
 - File
 - Continuation
 - File::Stat
 - Data
 - NilClass
 - Exception (see tree above)
 - Array
 - Proc
 - String
 - Numeric
 - Float
 - Integer
 - Bignum
 - Fixnum
 - Regexp
 - Thread
 - Module
 - Class
 - ThreadGroup
 - Method
 - UnboundMethod

- Struct
 - Struct::Tms
- TrueClass
- Time
- Dir
- Binding
- Range
- MatchData
- FalseClass

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 decode 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/*.rb 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 getopt
- pp prettier debugging output, 'p' on steroids.
- profile.rb ruby profiler - find that slow code!
- pstore.rb persistent object storage using marshal
- rexml/*.rb XML toolkit
- singleton.rb singleton design pattern library
- stringio lets you use an IO attached to a string.
- tempfile.rb temporary file that automatically removed
- test/unit unit testing framework
- 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

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.

```

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 specified 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 specified session
kill session  kills a specified session

```

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

xmp

```
require "irb/xmp"
xmp "something to eval" # or:
x = XMP.new
x.puts "something to eval"
```

ruby-mode

TODO: I don't have a freakin clue how to use the inferior ruby thing... I always fire up a shell in emacs... DOH!

Debugger

To invoke the debugger:

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

To use the debugger:

```
b[reak] [file:|class:]<line|method
b[reak] [class.]<line|method

wat[ch] expression          set breakpoint to some position
cat[ch] exception          set watchpoint to some expression
cat[ch]                    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
```

rdoc

```
=begin
the everything between a line beginning with `=begin' and
that with `=end' will be skipped by the interpreter.
=end
```

FIX: there is a lot more to rdoc.

Mindshare, Idiom and Patterns

Object Design

Visitor Pattern

By defining the method `#each` and including `Enumerable`, you get to use all the methods in `Enumerable`:

```
class Mailbox
  include Enumerable
  # ...
  def each
    @mail.each do
      # ...
      yield
    end
  end
end
```

Class SimpleDelegator, DelegateClass

```
foo = Object.new
foo2 = SimpleDelegator.new(foo)
foo.hash == foo2.hash # => false
Foo = DelegateClass(Array)
class ExtArray<DelegateClass(Array)
  ...
end
```

Module Observer

```
monitor.add_observer(self)
...
def update
  ...
  notify_observers(data, ...)
end
```

Module Singleton

```
class Klass
  include Singleton
  # ...
end

a, b = Klass.instance, Klass.instance
a == b # => true
a.new # raises NoMethodError
```

Other Third-party Libraries

Racc

- See i.loveruby.net/en/man/racc

Test::Unit

- `assert(boolean, message=nil)`
- `assert_block(message="assert_block failed.") do ... end`
- `assert_equal(expected, actual, message=nil)`
- `assert_in_delta(expected_float, actual_float, delta, message="")`
- `assert_instance_of(klass, object, message="")`
- `assert_kind_of(klass, object, message="")`
- `assert_match(pattern, string, message="")`
- `assert_nil(object, message="")`
- `assert_no_match(regexp, string, message="")`
- `assert_not_equal(expected, actual, message="")`
- `assert_not_nil(object, message="")`
- `assert_not_same(expected, actual, message="")`
- `assert_nothing_raised(*args)`
- `assert_nothing_thrown(message="") do ... end`
- `assert_operator(object1, operator, object2, message="")`

- `assert_raises(expected_exception_class, message="") do ... end`
- `assert_respond_to(object, method, message="")`
- `assert_same(expected, actual, message="")`
- `assert_send(send_array, message="")`
- `assert_throws(expected_symbol, message="") do ... end`
- `flunk(message="Flunked")`

Original URL: www.zenspider.com/Languages/Ruby/QuickRef.html
\$Author: ryand \$
\$Date: 2005/11/01 \$
\$Revision: #34 \$

[Sitemap](#) | [Search](#) || [Zen Spider Website](#) / [The Language Freak](#) / [Ruby](#) / [Ruby QuickRef](#)

"More matter, with less art" - Gertrude, Hamlet.

Copyright © 1997-2004 [Ryan Davis](#) & [Zen Spider Software](#). All Rights Reserved.

Generated: 2005-11-01 13:45:01