

NAME

ruby — Interpreted object-oriented scripting language

SYNOPSIS

```
ruby [ --copyright ] [ --version ] [ -SUacdlnpswvy ] [ -O[octal] ] [ -C directory ]
  [ -E external[:internal] ] [ -F[pattern] ] [ -I directory ] [ -K[c] ] [ -T[level] ]
  [ -W[level] ] [ -e command ] [ -i[extension] ] [ -r library ] [ -x[directory] ]
  [ -- { enable|disable }-FEATURE ] [ --dump=target ] [ --verbose ] [ -- ]
  [program_file] [argument . . .]
```

DESCRIPTION

Ruby is an interpreted scripting language for quick and easy object-oriented programming. It has many features to process text files and to do system management tasks (like in Perl). It is simple, straight-forward, and extensible.

If you want a language for easy object-oriented programming, or you don't like the Perl ugliness, or you do like the concept of LISP, but don't like too many parentheses, Ruby might be your language of choice.

FEATURES

Ruby's features are as follows:

Interpretive

Ruby is an interpreted language, so you don't have to recompile programs written in Ruby to execute them.

Variables have no type (dynamic typing)

Variables in Ruby can contain data of any type. You don't have to worry about variable typing. Consequently, it has a weaker compile time check.

No declaration needed

You can use variables in your Ruby programs without any declarations. Variable names denote their scope - global, class, instance, or local.

Simple syntax

Ruby has a simple syntax influenced slightly from Eiffel.

No user-level memory management

Ruby has automatic memory management. Objects no longer referenced from anywhere are automatically collected by the garbage collector built into the interpreter.

Everything is an object

Ruby is a purely object-oriented language, and was so since its creation. Even such basic data as integers are seen as objects.

Class, inheritance, and methods

Being an object-oriented language, Ruby naturally has basic features like classes, inheritance, and methods.

Singleton methods

Ruby has the ability to define methods for certain objects. For example, you can define a press-button action for certain widget by defining a singleton method for the button. Or, you can make up your own prototype based object system using singleton methods, if you want to.

Mix-in by modules

Ruby intentionally does not have the multiple inheritance as it is a source of confusion. Instead, Ruby has the ability to share implementations across the inheritance tree. This is often called a 'Mix-in'.

Iterators

Ruby has iterators for loop abstraction.

Closures

In Ruby, you can objectify the procedure.

Text processing and regular expressions

Ruby has a bunch of text processing features like in Perl.

M17N, character set independent

Ruby supports multilingualized programming. Easy to process texts written in many different natural languages and encoded in many different character encodings, without dependence on Unicode.

Bignums

With built-in bignums, you can for example calculate factorial(400).

Reflection and domain specific languages

Class is also an instance of the Class class. Definition of classes and methods is an expression just as 1+1 is. So your programs can even write and modify programs. Thus you can write your application in your own programming language on top of Ruby.

Exception handling

As in Java(tm).

Direct access to the OS

Ruby can use most UNIX system calls, often used in system programming.

Dynamic loading

On most UNIX systems, you can load object files into the Ruby interpreter on-the-fly.

Rich libraries

In addition to the “builtin libraries” and “standard libraries” that are bundled with Ruby, a vast amount of third-party libraries (“gems”) are available via the package management system called ‘RubyGems’, namely the `gem(1)` command. Visit RubyGems.org (<https://rubygems.org/>) to find the gems you need, and explore GitHub (<https://github.com/>) to see how they are being developed and used.

OPTIONS

The Ruby interpreter accepts the following command-line options (switches). They are quite similar to those of `perl(1)`.

- copyright** Prints the copyright notice, and quits immediately without running any script.
- version** Prints the version of the Ruby interpreter, and quits immediately without running any script.
- 0[*octal*]** (The digit “zero”.) Specifies the input record separator (`$/`) as an octal number. If no digit is given, the null character is taken as the separator. Other switches may follow the digits. **-00** turns Ruby into paragraph mode. **-0777** makes Ruby read whole file at once as a single string since there is no legal character with that value.
- C *directory***
- X *directory*** Causes Ruby to switch to the directory.
- E *external[:internal]***
- encoding *external[:internal]***
Specifies the default value(s) for external encodings and internal encoding. Values should be separated with colon (:).

You can omit the one for internal encodings, then the value (`Encoding.default_internal`) will be `nil`.

- external-encoding=encoding**
- internal-encoding=encoding**
Specify the default external or internal character encoding
- F pattern** Specifies input field separator (`$;`).
- I directory** Used to tell Ruby where to load the library scripts. Directory path will be added to the load-path variable (`$:`).
- K kcode** Specifies KANJI (Japanese) encoding. The default value for script encodings (`__ENCODING__`) and external encodings (`Encoding.default_external`) will be the specified one. *kcode* can be one of
 - e** EUC-JP
 - s** Windows-31J (CP932)
 - u** UTF-8
 - n** ASCII-8BIT (BINARY)
- S** Makes Ruby use the `PATH` environment variable to search for script, unless its name begins with a slash. This is used to emulate `#!` on machines that don't support it, in the following manner:


```
#! /usr/local/bin/ruby
# This line makes the next one a comment in Ruby \
exec /usr/local/bin/ruby -S $0 $*
```

On some systems `$0` does not always contain the full pathname, so you need the **-S** switch to tell Ruby to search for the script if necessary (to handle embedded spaces and such). A better construct than `$*` would be `${1+"$@"}`, but it does not work if the script is being interpreted by `csh(1)`.
- T[level=1]** Turns on taint checks at the specified level (default 1).
- U** Sets the default value for internal encodings (`Encoding.default_internal`) to UTF-8.
- W[level=2]** Turns on verbose mode at the specified level without printing the version message at the beginning. The level can be;
 - 0** Verbose mode is "silence". It sets the `$VERBOSE` to `nil`.
 - 1** Verbose mode is "medium". It sets the `$VERBOSE` to `false`.
 - 2 (default)** Verbose mode is "verbose". It sets the `$VERBOSE` to `true`. **-W2** is same as **-w**
- a** Turns on auto-split mode when used with **-n** or **-p**. In auto-split mode, Ruby executes `$F = $_.split` at beginning of each loop.
- c** Causes Ruby to check the syntax of the script and exit without executing. If there are no syntax errors, Ruby will print "Syntax OK" to the standard output.

- d**
--debug Turns on debug mode. `$DEBUG` will be set to true.
- e** *command* Specifies script from command-line while telling Ruby not to search the rest of the arguments for a script file name.
- h**
--help Prints a summary of the options.
- i** *extension* Specifies in-place-edit mode. The extension, if specified, is added to old file name to make a backup copy. For example:
- ```
% echo matz > /tmp/junk
% cat /tmp/junk
matz
% ruby -p -i.bak -e '$_.upcase!' /tmp/junk
% cat /tmp/junk
MATZ
% cat /tmp/junk.bak
matz
```
- l** (The lowercase letter “ell”.) Enables automatic line-ending processing, which means to firstly set `$\` to the value of `$/`, and secondly chops every line read using `chop!`.
- n** Causes Ruby to assume the following loop around your script, which makes it iterate over file name arguments somewhat like **sed -n** or **awk**.
- ```
while gets
  ...
end
```
- p** Acts mostly same as **-n** switch, but print the value of variable `$_` at the each end of the loop. For example:
- ```
% echo matz | ruby -p -e '$_.tr! "a-z", "A-Z"'
MATZ
```
- r** *library* Causes Ruby to load the library using `require`. It is useful when using **-n** or **-p**.
- s** Enables some switch parsing for switches after script name but before any file name arguments (or before a **--**). Any switches found there are removed from `ARGV` and set the corresponding variable in the script. For example:
- ```
#!/usr/local/bin/ruby -s
# prints "true" if invoked with '-xyz' switch.
print "true\n" if $xyz
```
- v** Enables verbose mode. Ruby will print its version at the beginning and set the variable `$VERBOSE` to true. Some methods print extra messages if this variable is true. If this switch is given, and no other switches are present, Ruby quits after printing its version.
- w** Enables verbose mode without printing version message at the beginning. It sets the `$VERBOSE` variable to true.
- x**[*directory*] Tells Ruby that the script is embedded in a message. Leading garbage will be discarded until the first line that starts with “#!” and contains the string, “ruby”. Any meaningful switches on that line will be applied. The end of the script must be specified with either EOF, `^D` (control-D), `^Z` (control-Z), or the reserved word `__END__`. If the directory name is specified, Ruby will switch to that directory before executing script.

-y
--yydebug DO NOT USE.

Turns on compiler debug mode. Ruby will print a bunch of internal state messages during compilation. Only specify this switch you are going to debug the Ruby interpreter.

--disable-*FEATURE*
--enable-*FEATURE*
 Disables (or enables) the specified *FEATURE*.

--disable-gems
--enable-gems Disables (or enables) RubyGems libraries. By default, Ruby will load the latest version of each installed gem. The Gem constant is true if RubyGems is enabled, false if otherwise.

--disable-rubyopt
--enable-rubyopt Ignores (or considers) the RUBYOPT environment variable. By default, Ruby considers the variable.

--disable-all
--enable-all Disables (or enables) all features.

--dump=*target*
 Dump some information.
 Prints the specified target. *target* can be one of;

- version** version description same as **--version**
- usage** brief usage message same as **-h**
- help** Show long help message same as **--help**
- syntax** check of syntax same as **-c --yydebug**
- yydebug** compiler debug mode, same as **--yydebug**
 Only specify this switch if you are going to debug the Ruby interpreter.
- parsetree**
parsetree_with_comment AST nodes tree
 Only specify this switch if you are going to debug the Ruby interpreter.
- insns** disassembled instructions
 Only specify this switch if you are going to debug the Ruby interpreter.

--verbose Enables verbose mode without printing version message at the beginning. It sets the \$VERBOSE variable to true. If this switch is given, and no script arguments (script file or **-e** options) are present, Ruby quits immediately.

ENVIRONMENT

RUBYLIB A colon-separated list of directories that are added to Ruby's library load path (\$:). Directories from this environment variable are searched before the standard load path is searched.

e.g.:

```
RUBYLIB="$HOME/lib/ruby:$HOME/lib/rubyext"
```

RUBYOPT Additional Ruby options.

e.g.

```
RUBYOPT="-w -Ke"
```

Note that RUBYOPT can contain only **-d**, **-E**, **-I**, **-K**, **-r**, **-T**, **-U**, **-v**, **-w**, **-W**, **--debug**, **--disable-*FEATURE*** and **--enable-*FEATURE***.

RUBYPATH A colon-separated list of directories that Ruby searches for Ruby programs when the **-S** flag is specified. This variable precedes the `PATH` environment variable.

RUBYSHELL The path to the system shell command. This environment variable is enabled for only `mswin32`, `mingw32`, and `OS/2` platforms. If this variable is not defined, Ruby refers to `COMSPEC`.

PATH Ruby refers to the `PATH` environment variable on calling `Kernel#system`.

And Ruby depends on some RubyGems related environment variables unless RubyGems is disabled. See the help of `gem(1)` as below.

```
% gem help
```

GC ENVIRONMENT

The Ruby garbage collector (GC) tracks objects in fixed-sized slots, but each object may have auxiliary memory allocations handled by the `malloc` family of C standard library calls (`malloc(3)`, `calloc(3)`, and `realloc(3)`). In this documentation, the "heap" refers to the Ruby object heap of fixed-sized slots, while "malloc" refers to auxiliary allocations commonly referred to as the "process heap". Thus there are at least two possible ways to trigger GC:

- 1 Reaching the object limit.
- 2 Reaching the malloc limit.

In Ruby 2.1, the generational GC was introduced and the limits are divided into young and old generations, providing two additional ways to trigger a GC:

- 3 Reaching the old object limit.
- 4 Reaching the old malloc limit.

There are currently 4 possible areas where the GC may be tuned by the following 11 environment variables:

RUBY_GC_HEAP_INIT_SLOTS	Initial allocation slots. Introduced in Ruby 2.1, default: 10000.
RUBY_GC_HEAP_FREE_SLOTS	Prepare at least this amount of slots after GC. Allocate this number slots if there are not enough slots. Introduced in Ruby 2.1, default: 4096
RUBY_GC_HEAP_GROWTH_FACTOR	Increase allocation rate of heap slots by this factor. Introduced in Ruby 2.1, default: 1.8, minimum: 1.0 (no growth)
RUBY_GC_HEAP_GROWTH_MAX_SLOTS	Allocation rate is limited to this number of slots, preventing excessive allocation due to RUBY_GC_HEAP_GROWTH_FACTOR. Introduced in Ruby 2.1, default: 0 (no limit)
RUBY_GC_HEAP_OLDOBJECT_LIMIT_FACTOR	Perform a full GC when the number of old objects is more than $R * N$, where R is this factor and N is the number of old objects after the last full GC. Introduced in Ruby 2.1.1, default: 2.0

RUBY_GC_MALLOC_LIMIT	The initial limit of young generation allocation from the malloc-family. GC will start when this limit is reached. Default: 16MB
RUBY_GC_MALLOC_LIMIT_MAX	The maximum limit of young generation allocation from malloc before GC starts. Prevents excessive malloc growth due to RUBY_GC_MALLOC_LIMIT_GROWTH_FACTOR. Introduced in Ruby 2.1, default: 32MB.
RUBY_GC_MALLOC_LIMIT_GROWTH_FACTOR	Increases the limit of young generation malloc calls, reducing GC frequency but increasing malloc growth until RUBY_GC_MALLOC_LIMIT_MAX is reached. Introduced in Ruby 2.1, default: 1.4, minimum: 1.0 (no growth)
RUBY_GC_OLDMALLOC_LIMIT	The initial limit of old generation allocation from malloc, a full GC will start when this limit is reached. Introduced in Ruby 2.1, default: 16MB
RUBY_GC_OLDMALLOC_LIMIT_MAX	The maximum limit of old generation allocation from malloc before a full GC starts. Prevents excessive malloc growth due to RUBY_GC_OLDMALLOC_LIMIT_GROWTH_FACTOR. Introduced in Ruby 2.1, default: 128MB
RUBY_GC_OLDMALLOC_LIMIT_GROWTH_FACTOR	Increases the limit of old generation malloc allocation, reducing full GC frequency but increasing malloc growth until RUBY_GC_OLDMALLOC_LIMIT_MAX is reached. Introduced in Ruby 2.1, default: 1.2, minimum: 1.0 (no growth)

STACK SIZE ENVIRONMENT

Stack size environment variables are implementation-dependent and subject to change with different versions of Ruby. The VM stack is used for pure-Ruby code and managed by the virtual machine. Machine stack is used by the operating system and its usage is dependent on C extensions as well as C compiler options. Using lower values for these may allow applications to keep more Fibers or Threads running; but increases the chance of SystemStackError exceptions and segmentation faults (SIGSEGV). These environment variables are available since Ruby 2.0.0. All values are specified in bytes.

RUBY_THREAD_VM_STACK_SIZE	VM stack size used at thread creation. default: 131072 (32-bit CPU) or 262144 (64-bit)
RUBY_THREAD_MACHINE_STACK_SIZE	Machine stack size used at thread creation. default: 524288 or 1048575
RUBY_FIBER_VM_STACK_SIZE	VM stack size used at fiber creation. default: 65536 or 131072
RUBY_FIBER_MACHINE_STACK_SIZE	Machine stack size used at fiber creation. default: 262144 or 524288

SEE ALSO

https://www.ruby-lang.org/	The official web site.
https://www.ruby-toolbox.com/	Comprehensive catalog of Ruby libraries.

REPORTING BUGS

- Security vulnerabilities should be reported via an email to `security@ruby-lang.org`. Reported problems will be published after being fixed.
- Other bugs and feature requests can be reported via the Ruby Issue Tracking System (<https://bugs.ruby-lang.org/>). Do not report security vulnerabilities via this system because it publishes the vulnerabilities immediately.

AUTHORS

Ruby is designed and implemented by Yukihiro Matsumoto <matz@netlab.jp>.

See <https://bugs.ruby-lang.org/projects/ruby/wiki/Contributors> for contributors to Ruby.