Ruby provides a whole set of I/O-related methods implemented in the Kernel module. All the I/O methods are derived from the class IO.

The class *IO* provides all the basic methods, such as *read, write, gets, puts, readline, getc,* and *printf*.

puts Statement

The *puts* statement instructs the program to display the value stored in the variable. This will add a new line at the end of each line it writes.

#!/usr/bin/ruby

val1 = "This is variable one"

val2 = "This is variable two"

puts val1

puts val2

## gets Statement

This code will prompt the user to enter a value, which will be stored in a variable val and finally will be printed on STDOUT.

#!/usr/bin/ruby

puts "Enter a value :"

val = gets

puts val

## putc Statement

Unlike the *puts* statement, which outputs the entire string onto the screen, the *putc* statement can be used to output one character at a time.

#!/usr/bin/ruby

str = "Hello Ruby!"

putc str

## print Statement

The *print* statement is similar to the *puts* statement. The only difference is that the *puts* statement goes to the next line after printing the contents, whereas with the *print* statement the cursor is positioned on the same line.

#!/usr/bin/ruby

print "Hello World"

print "Good Morning"

## Opening and Closing Files

|  |  |
| --- | --- |
| **Mode** | **Description** |
| R | Read only access. Pointer is positioned at start of file. |
| r+ | Read and write access. Pointer is positioned at start of file. |
| w | Write only access. Pointer is positioned at start of file. |
| w+ | Read and write access. Pointer is positioned at start of file. |
| a | Write only access. Pointer is positioned at end of file. |
| a+ | Read and write access. Pointer is positioned at end of file. |
| B | Binary File Mode. Used in conjunction with the above modes. Windows/DOS only. |
|  |  |

## File.new(“c://abcd/a.txt”) Method

You can create a *File* object using *File.new* method for reading, writing, or both, according to the mode string. Finally, you can use *File.close* method to close that file.

### Syntax

aFile = File.new("filename", "mode")

# ... process the file

aFile.close

## File.open Method

You can use *File.open* method to create a new file object and assign that file object to a file. However, there is one difference in between *File.open* and *File.new* methods. The difference is that the *File.open* method can be associated with a block, whereas you cannot do the same using the *File.new*method.

File.open("filename", "mode") do |aFile|

# ... process the file

end

## Reading and Writing Files

## he sysread Method

You can use the method *sysread* to read the contents of a file. You can open the file in any of the modes when using the method sysread. For example −

Following is the input text file −

This is a simple text file for testing purpose.

Now let's try to read this file −

#!/usr/bin/ruby

aFile = File.new("input.txt", "r")

if aFile

content = aFile.sysread(20) # 20 chars

puts content

else

puts "Unable to open file!"

end

This statement will output the first 20 characters of the file. The file pointer will now be placed at the 21st character in the file.

## The syswrite Method

You can use the method syswrite to write the contents into a file. You need to open the file in write mode when using the method syswrite. For example −

#!/usr/bin/ruby

aFile = File.new("input.txt", "r+")

if aFile

aFile.syswrite("ABCDEF")

else

puts "Unable to open file!"

end

This statement will write "ABCDEF" into the file.

## The each\_byte Method

This method belongs to the class *File*. The method *each\_byte* is always associated with a block. Consider the following code sample −

#!/usr/bin/ruby

aFile = File.new("input.txt", "r+")

if aFile

aFile.syswrite("ABCDEF")

aFile.each\_byte {|ch| putc ch; putc ?. }

else

puts "Unable to open file!"

end

## The IO.readlines Method

The class *File* is a subclass of the class IO. The class IO also has some methods, which can be used to manipulate files.

One of the IO class methods is *IO.readlines*. This method returns the contents of the file line by line. The following code displays the use of the method *IO.readlines* −

#!/usr/bin/ruby

arr = IO.readlines("input.txt")

puts arr[0]

puts arr[1]

## The IO.foreach Method

This method also returns output line by line. The difference between the method *foreach* and the method *readlines* is that the method *foreach* is associated with a block. However, unlike the method *readlines*, the method *foreach* does not return an array. For example −

#!/usr/bin/ruby

IO.foreach("input.txt"){|block| puts block}

This code will pass the contents of the file *test* line by line to the variable block, and then the output will be displayed on the screen.

## Renaming and Deleting Files

You can rename and delete files programmatically with Ruby with the *rename*and *delete* methods.

Following is the example to rename an existing file *test1.txt* −

#!/usr/bin/ruby

# Rename a file from test1.txt to test2.txt

File.rename( "test1.txt", "test2.txt" )

Following is the example to delete an existing file *test2.txt* −

#!/usr/bin/ruby

# Delete file test2.txt

File.delete("test2.txt")

## File Modes and Ownership

Use the *chmod* method with a mask to change the mode or permissions/access list of a file −

Following is the example to change mode of an existing file *test.txt* to a mask value −

#!/usr/bin/ruby

file = File.new( "test.txt", "w" )

file.chmod( 0755 )

## File Inquiries

The following command tests whether a file exists before opening it −

#!/usr/bin/ruby

File.open("file.rb") if File::exists?( "file.rb" )

The following command inquire whether the file is really a file −

#!/usr/bin/ruby

# This returns either true or false

File.file?( "text.txt" )

The following command finds out if the given file name is a directory −

#!/usr/bin/ruby

# a directory

File::directory?( "/usr/local/bin" ) # => true

# a file

File::directory?( "file.rb" ) # => false

The following command finds whether the file is readable, writable or executable −

#!/usr/bin/ruby

File.readable?( "test.txt" ) # => true

File.writable?( "test.txt" ) # => true

File.executable?( "test.txt" ) # => false

The following command finds whether the file has zero size or not −

#!/usr/bin/ruby

File.zero?( "test.txt" ) # => true

The following command returns size of the file −

#!/usr/bin/ruby

File.size?( "text.txt" ) # => 1002

The following command can be used to find out a type of file −

#!/usr/bin/ruby

File::ftype( "test.txt" ) # => file

The ftype method identifies the type of the file by returning one of the following − *file, directory, characterSpecial, blockSpecial, fifo, link, socket, or unknown.*

The following command can be used to find when a file was created, modified, or last accessed −

#!/usr/bin/ruby

File::ctime( "test.txt" ) # => Fri May 09 10:06:37 -0700 2008

File::mtime( "text.txt" ) # => Fri May 09 10:44:44 -0700 2008

File::atime( "text.txt" ) # => Fri May 09 10:45:01 -0700 2008

## Directories in Ruby

All files are contained within various directories, and Ruby has no problem handling these too. Whereas the *File* class handles files, directories are handled with the *Dir* class.

## Navigating Through Directories

To change directory within a Ruby program, use *Dir.chdir* as follows. This example changes the current directory to */usr/bin*.

Dir.chdir("/usr/bin")

You can find out what the current directory is with *Dir.pwd* −

puts Dir.pwd # This will return something like /usr/bin

You can get a list of the files and directories within a specific directory using *Dir.entries* −

puts Dir.entries("/usr/bin").join(' ')

*Dir.entries* returns an array with all the entries within the specified directory. *Dir.foreach* provides the same feature −

Dir.foreach("/usr/bin") do |entry|

puts entry

end

An even more concise way of getting directory listings is by using Dir's class array method −

Dir["/usr/bin/\*"]

## Creating a Directory

The *Dir.mkdir* can be used to create directories −

Dir.mkdir("mynewdir")

You can also set permissions on a new directory (not one that already exists) with mkdir −

**NOTE** − The mask 755 sets permissions owner, group, world [anyone] to rwxr-xr-x where r = read, w = write, and x = execute.

Dir.mkdir( "mynewdir", 755 )

## Deleting a Directory

The *Dir.delete* can be used to delete a directory. The *Dir.unlink* and *Dir.rmdir*performs exactly the same function and are provided for convenience.

Dir.delete("testdir")

## Creating Files & Temporary Directories

Temporary files are those that might be created briefly during a program's execution but aren't a permanent store of information.

*Dir.tmpdir* provides the path to the temporary directory on the current system, although the method is not available by default. To make *Dir.tmpdir*available it's necessary to use require 'tmpdir'.

You can use *Dir.tmpdir* with *File.join* to create a platform-independent temporary file −

require 'tmpdir'

tempfilename = File.join(Dir.tmpdir, "tingtong")

tempfile = File.new(tempfilename, "w")

tempfile.puts "This is a temporary file"

tempfile.close

File.delete(tempfilename)

This code creates a temporary file, writes data to it, and deletes it. Ruby's standard library also includes a library called *Tempfile* that can create temporary files for you −

require 'tempfile'

f = Tempfile.new('tingtong')

f.puts "Hello"

puts f.path

f.close

Read All files from directory

Dir["/path/to/search/\*"]

Dir["/path/to/search/\*\*/\*.rb"]

You can use this:

files = Dir.foreach(dir).select { |x| File.file?("#{dir}/#{x}") }

This returns the filenames, i.e. without folder.

If you need the complete path, use something like this:

files = Dir.foreach(dir) \

.map { |x| File.expand\_path("#{dir}/#{x}") } \

.select { |x| File.file?(x) }

$ **irb**

>> **basedir = '.'**

=> "."

>> **files = Dir.glob("\*.jpg")**

=> ["disk-inventory-x-1.jpg", "disk-inventory-x-2.jpg", "omni-disk-sweeper-1.jpg"]

>> **puts files**

disk-inventory-x-1.jpg

disk-inventory-x-2.jpg

omni-disk-sweeper-1.jpg

=> nil