Class File < IO

A File is an abstraction of any file object accessible by the program and is closely associated with class IO, page 546. File includes the methods of module FileTest as class methods, allowing you to write (for example) File.exist?("foo").

In this section, *permission bits* are a platform-specific set of bits that indicate permissions of a file. On Unix-based systems, permissions are viewed as a set of three octets, for the owner, the group, and the rest of the world. For each of these entities, permissions may be set to read, write, or execute the file.

Owner			Group			Other		
r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1

The permission bits 0644 (in octal) would thus be interpreted as read/write for owner and read-only for group and other. Higher-order bits may also be used to indicate the type of file (plain, directory, pipe, socket, and so on) and various other special features. If the permissions are for a directory, the meaning of the execute bit changes; when set, the directory can be searched.

Each file has three associated times. The *atime* is the time the file was last accessed. The *ctime* is the time that the file status (not necessarily the file contents) were last changed. Finally, the *mtime* is the time the file's data was last modified. In Ruby, all these times are returned as Time objects.

On non-POSIX operating systems, there may be only the ability to make a file read-only or read/write. In this case, the remaining permission bits will be synthesized to resemble typical values. For instance, on Windows the default permission bits are 0644, which means read/write for owner and read-only for all others. The only change that can be made is to make the file read-only, which is reported as 0444.

See also Pathname on page 788 and IO on page 546.

Class methods

absolute_path File.absolute_path(*filename* ⟨ , *dirstring* ⟩) → *filename*

1.9

Converts a path name to an absolute path name. Relative paths are referenced from the current working directory of the process unless *dirstring* is given, in which case it will be used as the starting point. Path names starting with ~ are not expanded, in contrast with File#expand_path.

```
puts File.absolute_path("bin")
puts File.absolute_path("../../bin", "/tmp/x")
```

produces:

```
/Users/dave/BS2/titles/RUBY3/Book/bin
/bin
```

atime File.atime(*filename*) → *time*

Returns a Time object containing the last access time for the named file, or returns epoch if the file has not been accessed.

```
File.atime("testfile")   # =>   2009-04-13 13:26:21 -0500
```

basename File.basename(*filename* ⟨ , *suffix* ⟩) → *string*

Returns the last component of the filename given in *filename*. If *suffix* is given and is present at the end of *filename*, it is removed. Any extension can be removed by giving an extension of .*.

```
File.basename("/home/gumby/work/ruby.rb")          # =>   "ruby.rb"
File.basename("/home/gumby/work/ruby.rb", ".rb")   # =>   "ruby"
File.basename("/home/gumby/work/ruby.rb", ".*")    # =>   "ruby"
```

blockdev? File.blockdev?(*filename*) → true or false

Returns true if the named file is a block device and returns false if it isn't or if the operating system doesn't support this feature.

```
File.blockdev?("testfile")   # =>   false
```

chardev? File.chardev?(*filename*) → true or false

Returns true if the named file is a character device and returns false if it isn't or if the operating system doesn't support this feature.

```
File.chardev?("/dev/tty")   # =>   true
```

chmod File.chmod(*permission* ⟨ , *filename* ⟩+) → *int*

Changes permission bits on the named file(s) to the bit pattern represented by *permission*. Actual effects are operating system dependent (see the beginning of this section). On Unix systems, see chmod(2) for details. Returns the number of files processed.

```
File.chmod(0644, "testfile", "out")   # =>   2
```

chown File.chown(*owner*, *group* ⟨ , *filename* ⟩+) → *int*

Changes the owner and/or group of the named file(s) to the given numeric owner and group IDs. Only a process with superuser privileges may change the owner of a file. The current owner of a file may change the file's group to any group to which the owner belongs. A nil or −1 owner or group ID is ignored. Returns the number of files processed.

```
File.chown(nil, 100, "testfile")
```

ctime File.ctime(*filename*) → *time*

Returns a Time object containing the time that the file status associated with the named file was changed.

```
File.ctime("testfile")   # =>   2009-04-13 13:26:22 -0500
```

delete File.delete(⟨ *filename* ⟩+) → *int*

Deletes the named file(s). Returns the number of files processed. See also Dir.rmdir.

```
File.open("testrm", "w+") {}
File.delete("testrm")   # =>   1
```

directory? File.directory?(*path*) → true or false

Returns true if the named file is a directory; returns false otherwise.

```
File.directory?(".")   # =>   true
```

dirname File.dirname(*filename*) → *filename*

Returns all components of the filename given in *filename* except the last one.

```
File.dirname("/home/gumby/work/ruby.rb")   # =>   "/home/gumby/work"
File.dirname("ruby.rb")                    # =>   "."
```

executable? File.executable?(*filename*) → true or false

Returns true if the named file is executable. The tests are made using the effective owner of the process.

```
File.executable?("testfile")   # =>   false
```

executable_real? File.executable_real?(*filename*) → true or false

Same as File#executable? but tests using the real owner of the process.

exist? File.exist?(*filename*) → true or false

Returns true if the named file or directory exists.

```
File.exist?("testfile")   # =>   true
```

exists? File.exists? (*filename*) → true or false

Synonym for File.exist?.

expand_path File.expand_path(*filename* ⟨ , *dirstring* ⟩) → *filename*

Converts a path name to an absolute path name. Relative paths are referenced from the current working directory of the process unless *dirstring* is given, in which case it will be used as the starting point. The given path name may start with a ~, which expands to the process owner's home directory (the environment variable HOME must be set correctly). ~*user* expands to the named user's home directory. See also File#absolute_path.

```
File.expand_path("~/bin")                 # =>   "/Users/dave/bin"
File.expand_path("../../bin", "/tmp/x")   # =>   "/bin"
```

extname File.extname(*path*) → *string*

Returns the extension (the portion of filename in *path* after the period).

```
File.extname("test.rb")         # =>   ".rb"
File.extname("a/b/d/test.rb")   # =>   ".rb"
File.extname("test")            # =>   ""
```

file?

File.file?(*filename*) → true or false

Returns true if the named file is a regular file (not a device file, directory, pipe, socket, and so on).

```
File.file?("testfile")   # =>   true
File.file?(".")          # =>   false
```

fnmatch

File.fnmatch(*glob_pattern*, *path*, ⟨ *flags* ⟩) → true or false

1.9

Returns true if *path* matches against *glob_pattern*. (As of Ruby 1.9, File.fnmatch is an alias for Dir.fnmatch.) The pattern is not a regular expression; instead, it follows rules similar to shell filename globbing. A *glob_pattern* may contain the following metacharacters.

*	Matches zero or more characters in a file or directory name.
**	Matches zero or more characters, ignoring name boundaries. Most often used to scan subdirectories recursively.
?	Matches any single character.
[*charset*]	Matches any character from the given set of characters. A range of characters is written as *from-to*. The set may be negated with an initial caret (^).
\	Escapes any special meaning of the next character.

flags is a bitwise OR of the FNM_xxx parameters listed on the next page. See also Dir.glob on page 480.

```
File.fnmatch('cat',       'cat')        # =>   true
File.fnmatch('cat',       'category')   # =>   false
File.fnmatch('c?t',       'cat')        # =>   true
File.fnmatch('c\?t',      'cat')        # =>   false

File.fnmatch('c??t',   'cat')                     # =>   false
File.fnmatch('c*',     'cats')                    # =>   true
File.fnmatch('c/**/t', 'c/a/b/c/t')               # =>   true
File.fnmatch('c**t',   'c/a/b/c/t')               # =>   true
File.fnmatch('c**t',   'cat')                     # =>   true
File.fnmatch('**.txt', 'notes.txt')               # =>   true
File.fnmatch('**.txt', 'some/dir/tree/notes.txt') # =>   true
File.fnmatch('c*t',    'cat')                     # =>   true
File.fnmatch('c\at',   'cat')                     # =>   true
File.fnmatch('c\at',   'cat', File::FNM_NOESCAPE) # =>   false
File.fnmatch('a?b',    'a/b')                     # =>   true
File.fnmatch('a?b',    'a/b', File::FNM_PATHNAME) # =>   false
```

Table 27.3. Match-Mode Constants

FNM_NOESCAPE	Backslash does not escape special characters in globs, and a backslash in the pattern must match a backslash in the filename.
FNM_PATHNAME	Forward slashes in the filename are treated as separating parts of a path and so must be explicitly matched in the pattern.
FNM_DOTMATCH	If this option is not specified, filenames containing leading periods must be matched by an explicit period in the pattern. A leading period is one at the start of the filename or (if FNM_PATHNAME is specified) following a slash.
FNM_CASEFOLD	Filename matches are case insensitive

```
File.fnmatch('*',   '.profile')                          # =>   false
File.fnmatch('*',   '.profile', File::FNM_DOTMATCH)      # =>   true
File.fnmatch('*',   'dave/.profile')                     # =>   true
File.fnmatch('*',   'dave/.profile', File::FNM_DOTMATCH) # =>   true
File.fnmatch('*',   'dave/.profile', File::FNM_PATHNAME) # =>   false
File.fnmatch('*/*', 'dave/.profile', File::FNM_PATHNAME) # =>   false
STRICT = File::FNM_PATHNAME | File::FNM_DOTMATCH
File.fnmatch('*/*', 'dave/.profile', STRICT)             # =>   true
```

fnmatch?

File.fnmatch?(*glob_pattern*, *path*, ⟨ *flags* ⟩) → (true or false)

Synonym for File#fnmatch.

ftype

File.ftype(*filename*) → *filetype*

Identifies the type of the named file. The return string is one of file, directory, characterSpecial, blockSpecial, fifo, link, socket, or unknown.

```
File.ftype("testfile")    # =>   "file"
File.ftype("/dev/tty")    # =>   "characterSpecial"
system("mkfifo wibble")   # =>   true
File.ftype("wibble")      # =>   "fifo"
```

grpowned?

File.grpowned?(*filename*) → true or false

Returns true if the effective group ID of the process is the same as the group ID of the named file. On Windows, returns false.

```
File.grpowned?("/etc/passwd")   # =>   false
```

identical?

File.identical?(*name1*, *name2*) → true or false

1.9

Returns true only if *name1* and *name2* refer to the same file. Two separate files with the same content are not considered to be identical.

```
File.identical?("testfile", "./code/../testfile")   # =>   true
File.symlink("testfile", "wibble")
File.identical?("testfile", "wibble")               # =>   true
File.link("testfile", "wobble")
File.identical?("testfile", "wobble")               # =>   true
File.identical?("wibble", "wobble")                 # =>   true
```

join

File.join(⟨ *string* ⟩+) → *filename*

Returns a new string formed by joining the strings using File::SEPARATOR. The various separators are listed in Table 27.4 on the next page.

```
File.join("usr", "mail", "gumby")   # =>   "usr/mail/gumby"
```

lchmod

File.lchmod(*permission*, ⟨ *filename* ⟩+) → 0

Equivalent to File.chmod but does not follow symbolic links (so it will change the permissions associated with the link, not the file referenced by the link). Often not available.

lchown

File.lchown(*owner*, *group*, ⟨ *filename* ⟩+) → 0

Equivalent to File.chown but does not follow symbolic links (so it will change the owner associated with the link, not the file referenced by the link). Often not available.

link

File.link(*oldname*, *newname*) → 0

Creates a new name for an existing file using a hard link. Will not overwrite *newname* if it already exists (in which case link raises a subclass of SystemCallError). Not available on all platforms.

```
File.link("testfile", "testfile.2")   # =>   0
f = File.open("testfile.2")
f.gets                                # =>   "This is line one\n"
File.delete("testfile.2")
```

lstat

File.lstat(*filename*) → *stat*

Returns status information for *file* as an object of type File::Stat. Same as IO#stat (see page 561), but does not follow the last symbolic link. Instead, reports on the link itself.

```
File.symlink("testfile", "link2test")   # =>   0
File.stat("testfile").size              # =>   66
File.lstat("link2test").size            # =>   8
File.stat("link2test").size             # =>   66
```

mtime

File.mtime(*filename*) → *time*

Returns a Time object containing the modification time for the named file.

```
File.mtime("testfile")   # =>   2009-04-13 12:45:10 -0500
File.mtime("/tmp")       # =>   2009-04-13 13:16:08 -0500
```

Table 27.4. Path Separator Constants (Platform-Specific)

ALT_SEPARATOR	Alternate path separator (\ on Windows, nil otherwise).
PATH_SEPARATOR	Separator for filenames in a search path (such as : or ;).
SEPARATOR	Separator for directory components in a filename (such as \ or /).
Separator	Alias for SEPARATOR.

new File.new(*filename*, *modestring*="r") → *file*
File.new(*filename* ⟨ , *modenum* ⟨ , *permission* ⟩ ⟩) → *file*
File.new(*fd* ⟨ , *modenum* ⟨ , *permission* ⟩ ⟩) → *file*

1.9

Opens the file named by *filename* according to *modestring* (the default is "r") and returns a new File object. The *modestring* is described in Table 27.7 on page 547—it contains both information on the way the file is to be opened and optionally on the encodings to be associated with the file data. The file mode may optionally be specified as a Fixnum by *or*-ing together the flags described in Table 27.5 on page 514. Optional permission bits may be given in *permission*. These mode and permission bits are platform dependent; on Unix systems, see open(2) for details. If the first parameter is an integer (or can be converted to an integer using to_int, it is assumed to be the file descriptor or an already-open file. In that case, the call is passed to IO.new for processing. See also IO.open on page 514 for a block form of File.new.

```
# open for reading, default external encoding
f = File.new("testfile", "r")
# open for reading, assume contents are utf-8
f = File.new("testfile", "r:utf-8")
# open for read/write. external utf-8 data will be converted to iso-8859-1
# when read, and converted from 8859-1 to utf-8 on writing
f = File.new("newfile",  "w+:utf-8:iso-8859-1")
# same as specifying "w+"
f = File.new("newfile", File::CREAT|File::TRUNC|File::RDWR, 0644)
```

owned? File.owned?(*filename*) → true or false

Returns true if the effective user ID of the process is the same as the owner of the named file.

```
File.owned?("/etc/passwd")   # =>   false
```

path File.path(*obj*) → *string*

1.9

Returns the path of *obj*. If *obj* responds to to_path, its value is returned. Otherwise, attempt to convert *obj* to a string and return that value.

```
File.path("testfile")          # =>   "testfile"
File.path("/tmp/../tmp/xxx")   # =>   "/tmp/../tmp/xxx"
f = File.open("/tmp/../tmp/xxx")
File.path(f)                   # =>   "/tmp/../tmp/xxx"
```

pipe? File.pipe?(*filename*) → true or false

Returns true if the OS supports pipes and the named file is one; false otherwise.

```
File.pipe?("testfile")   # =>   false
```

readable? File.readable?(*filename*) → true or false

Returns true if the named file is readable by the effective user ID of this process.

```
File.readable?("testfile")   # =>   true
```

readable_real? File.readable_real?(*filename*) → true or false

Returns true if the named file is readable by the real user ID of this process.

```
File.readable_real?("testfile")   # =>   true
```

readlink File.readlink(*filename*) → *filename*

Returns the given symbolic link as a string. Not available on all platforms.

```
File.symlink("testfile", "link2test")   # =>   0
File.readlink("link2test")              # =>   "testfile"
```

rename File.rename(*oldname*, *newname*) → 0

Renames the given file or directory to the new name. Raises a SystemCallError if the file cannot be renamed.

```
File.rename("afile", "afile.bak")   # =>   0
```

setgid? File.setgid?(*filename*) → true or false

Returns true if the named file's set-group-id permission bit is set and returns false if it isn't or if the operating system doesn't support this feature.

```
File.setgid?("/usr/sbin/lpc")   # =>   false
```

setuid? File.setuid?(*filename*) → true or false

Returns true if the named file's set-user-id permission bit is set and returns false if it isn't or if the operating system doesn't support this feature.

```
File.setuid?("/bin/su")   # =>   false
```

size File.size(*filename*) → *int*

Returns the size of the file in bytes.

```
File.size("testfile")   # =>   66
```

size? File.size?(*filename*) → *int* or nil

Returns nil if the named file is of zero length; otherwise, returns the size. Usable as a condition in tests.

```
File.size?("testfile")    # =>   66
File.size?("/dev/zero")   # =>   nil
```

Table 27.5. Open-Mode Constants

File::APPEND	Opens the file in append mode; all writes will occur at end of file.
File::CREAT	Creates the file on open if it does not exist.
File::EXCL	When used with File::CREAT, opens will fail if the file exists.
File::NOCTTY	When opening a terminal device (see IO#isatty on page 557), does not allow it to become the controlling terminal.
File::NONBLOCK	Opens the file in nonblocking mode.
File::RDONLY	Opens for reading only.
File::RDWR	Opens for reading and writing.
File::TRUNC	Opens the file and truncates it to zero length if the file exists.
File::WRONLY	Opens for writing only.

socket?

File.socket?(*filename*) → true or false

Returns true if the named file is a socket and returns false if it isn't or if the operating system doesn't support this feature.

split

File.split(*filename*) → *array*

Splits the given string into a directory and a file component and returns them in a two-element array. See also File.dirname and File.basename.

```
File.split("/home/gumby/.profile")   # =>   ["/home/gumby", ".profile"]
File.split("ruby.rb")                # =>   [".", "ruby.rb"]
```

stat

File.stat(*filename*) → *stat*

Returns a File::Stat object for the named file (see File::Stat, page 518).

```
stat = File.stat("testfile")
stat.mtime   # =>   2009-04-13 12:45:10 -0500
stat.ftype   # =>   "file"
```

sticky?

File.sticky?(*filename*) → true or false

Returns true if the named file has its sticky bit set and returns false if it doesn't or if the operating system doesn't support this feature.

symlink

File.symlink(*oldname*, *newname*) → 0 or nil

Creates a symbolic link called *newname* for the file *oldname*. Returns nil on all platforms that do not support symbolic links.

```
File.symlink("testfile", "link2test")   # =>   0
```

symlink?

File.symlink?(*filename*) → true or false

Returns true if the named file is a symbolic link and returns false if it isn't or if the operating system doesn't support this feature.

```
File.symlink("testfile", "link2test")   # =>   0
File.symlink?("link2test")              # =>   true
```

truncate File.truncate(*filename*, *int*) → 0

Truncates the file *filename* to be at most *int* bytes long. Not available on all platforms.

```
f = File.new("out", "w")
f.write("1234567890")     # =>   10
f.close                   # =>   nil
File.truncate("out", 5)   # =>   0
File.size("out")          # =>   5
```

umask File.umask(⟨ *int* ⟩) → *int*

Returns the current umask value for this process. If the optional argument is given, sets the umask to that value and returns the previous value. Umask values are *excluded* from the default permissions; so a umask of 0222 would make a file read-only for everyone. See also the discussion of permissions on page 506.

```
File.umask(0006)   # =>   18
File.umask         # =>   6
```

unlink File.unlink(⟨ *filename* ⟩+) → *int*

Synonym for File.delete. See also Dir.rmdir.

```
File.open("testrm", "w+") {}   # =>   nil
File.unlink("testrm")          # =>   1
```

utime File.utime(*accesstime*, *modtime* ⟨ , *filename* ⟩+) → *int*

Changes the access and modification times on a number of files. The times must be instances of class Time or integers representing the number of seconds since epoch. Returns the number of files processed. Not available on all platforms.

```
File.utime(0, 0, "testfile")          # =>   1
File.mtime("testfile")                # =>   1969-12-31 18:00:00 -0600
File.utime(0, Time.now, "testfile")   # =>   1
File.mtime("testfile")                # =>   2009-04-13 13:26:23 -0500
```

world_readable? File.world_readable?(*filename*) → *perm_int* or nil

1.9

If *filename* is readable by others, returns an integer representing the file permission bits of *filename*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

```
File.world_readable?("/etc/passwd")           # =>   420
File.world_readable?("/etc/passwd").to_s(8)   # =>   "644"
```

world_writable? File.world_writable?(*filename*) → *perm_int* or nil

1.9

If *filename* is writable by others, returns an integer representing the file permission bits of *filename*. Returns nil otherwise. The meaning of the bits is platform dependent; on Unix systems, see stat(2).

```
File.world_writable?("/etc/passwd")    # =>   nil
File.world_writable?("/tmp")           # =>   511
File.world_writable?("/tmp").to_s(8)   # =>   "777"
```

writable? File.writable?(*filename*) → true or false

Returns true if the named file is writable by the effective user ID of this process.

```
File.writable?("/etc/passwd")   # =>   false
File.writable?("testfile")      # =>   true
```

writable_real? File.writable_real?(*filename*) → true or false

Returns true if the named file is writable by the real user ID of this process.

zero? File.zero?(*filename*) → true or false

Returns true if the named file is of zero length and returns false otherwise.

```
File.zero?("testfile")   # =>   false
File.open("zerosize", "w") {}
File.zero?("zerosize")   # =>   true
```

Instance methods

atime *file*.atime → *time*

Returns a Time object containing the last access time for *file*, or returns epoch if the file has not been accessed.

```
File.new("testfile").atime   # =>   1969-12-31 18:00:00 -0600
```

chmod *file*.chmod(*permission*) → 0

Changes permission bits on *file* to the bit pattern represented by *permission*. Actual effects are platform dependent; on Unix systems, see chmod(2) for details. Follows symbolic links. See the discussion of permissions on page 506. Also see File#lchmod.

```
f = File.new("out", "w");
f.chmod(0644)   # =>   0
```

chown *file*.chown(*owner*, *group*) → 0

Changes the owner and group of *file* to the given numeric owner and group IDs. Only a process with superuser privileges may change the owner of a file. The current owner of a file may change the file's group to any group to which the owner belongs. A nil or −1 owner or group id is ignored. Follows symbolic links. See also File#lchown.

```
File.new("testfile").chown(502, 1000)
```

ctime *file*.ctime → *time*

Returns a Time object containing the time that the file status associated with *file* was changed.

```
File.new("testfile").ctime   # =>   2009-04-13 13:26:23 -0500
```

flock

file.flock (*locking_constant*) → 0 or false

Locks or unlocks a file according to *locking_constant* (a logical *or* of the values shown in Table 27.6 on the following page). Returns false if File::LOCK_NB is specified and the operation would otherwise have blocked. Not available on all platforms.

```
File.new("testfile").flock(File::LOCK_UN)   # =>   0
```

lchmod

file.lchmod(*permission*) → 0

Equivalent to File#chmod but does not follow symbolic links (so it will change the permissions associated with the link, not the file referenced by the link). Often not available.

lchown

file.lchown(*owner*, *group*) → 0

Equivalent to File#chown but does not follow symbolic links (so it will change the owner associated with the link, not the file referenced by the link). Often not available.

lstat

file.lstat → *stat*

Same as IO#stat but does not follow the last symbolic link. Instead, reports on the link itself.

```
File.symlink("testfile", "link2test")   # =>   0
File.stat("testfile").size              # =>   66
f = File.new("link2test")
f.lstat.size                            # =>   8
f.stat.size                             # =>   66
```

mtime

file.mtime → *time*

Returns a Time object containing the modification time for *file*.

```
File.new("testfile").mtime   # =>   2009-04-13 13:26:23 -0500
```

path

file.path → *filename*

Returns the path name used to create *file* as a string. Does not normalize the name.

```
File.new("testfile").path               # =>   "testfile"
File.new("/tmp/../tmp/xxx", "w").path   # =>   "/tmp/../tmp/xxx"
```

to_path

file.to_path → *filename*

Alias for File#path.

truncate

file.truncate(*int*) → 0

Truncates *file* to at most *int* bytes. The file must be opened for writing. Not available on all platforms.

```
f = File.new("out", "w")
f.syswrite("1234567890")   # =>   10
f.truncate(5)              # =>   0
f.close()                  # =>   nil
File.size("out")           # =>   5
```