## NAME

File::Copy - Copy files or filehandles

## SYNOPSIS

```
use File::Copy;

copy("file1","file2") or die "Copy failed: $!";
copy("Copy.pm",\*STDOUT);
move("/dev1/fileA","/dev2/fileB");

use File::Copy "cp";

$n = FileHandle->new("/a/file","r");
cp($n,"x");
```

## DESCRIPTION

The File::Copy module provides two basic functions, `copy` and `move`, which are useful for getting the contents of a file from one place to another.

copy

The `copy` function takes two parameters: a file to copy from and a file to copy to. Either argument may be a string, a FileHandle reference or a FileHandle glob. Obviously, if the first argument is a filehandle of some sort, it will be read from, and if it is a file *name* it will be opened for reading. Likewise, the second argument will be written to (and created if need be). Trying to copy a file on top of itself is a fatal error.

If the destination (second argument) already exists and is a directory, and the source (first argument) is not a filehandle, then the source file will be copied into the directory specified by the destination, using the same base name as the source file. It's a failure to have a filehandle as the source when the destination is a directory.

**Note that passing in files as handles instead of names may lead to loss of information on some operating systems; it is recommended that you use file names whenever possible.** Files are opened in binary mode where applicable. To get a consistent behaviour when copying from a filehandle to a file, use `binmode` on the filehandle.

An optional third parameter can be used to specify the buffer size used for copying. This is the number of bytes from the first file, that will be held in memory at any given time, before being written to the second file. The default buffer size depends upon the file, but will generally be the whole file (up to 2MB), or 1k for filehandles that do not reference files (eg. sockets).

You may use the syntax `use File::Copy "cp"` to get at the `cp` alias for this function. The syntax is *exactly* the same. The behavior is nearly the same as well: as of version 2.15, <cp> will preserve the source file's permission bits like the shell utility `cp(1)` would do, while `copy` uses the default permissions for the target file (which may depend on the process' `umask`, file ownership, inherited ACLs, etc.). If an error occurs in setting permissions, `cp` will return 0, regardless of whether the file was successfully copied.

move

The `move` function also takes two parameters: the current name and the intended name of the file to be moved. If the destination already exists and is a directory, and the source is not a directory, then the source file will be renamed into the directory specified by the destination.

If possible, move() will simply rename the file. Otherwise, it copies the file to the new location and deletes the original. If an error occurs during this copy-and-delete process, you may be left with a (possibly partial) copy of the file under the destination name.

You may use the `mv` alias for this function in the same way that you may use the <cp> alias for

copy.

syscopy

File::Copy also provides the `syscopy` routine, which copies the file specified in the first parameter to the file specified in the second parameter, preserving OS-specific attributes and file structure. For Unix systems, this is equivalent to the simple `copy` routine, which doesn't preserve OS-specific attributes. For VMS systems, this calls the `rmscopy` routine (see below). For OS/2 systems, this calls the `syscopy` XSUB directly. For Win32 systems, this calls `Win32::CopyFile`.

On Mac OS (Classic), `syscopy` calls `Mac::MoreFiles::FSpFileCopy`, if available.

**Special behaviour if syscopy is defined (OS/2, VMS and Win32)**:

If both arguments to `copy` are not file handles, then `copy` will perform a "system copy" of the input file to a new output file, in order to preserve file attributes, indexed file structure, *etc.* The buffer size parameter is ignored. If either argument to `copy` is a handle to an opened file, then data is copied using Perl operators, and no effort is made to preserve file attributes or record structure.

The system copy routine may also be called directly under VMS and OS/2 as `File::Copy::syscopy` (or under VMS as `File::Copy::rmscopy`, which is the routine that does the actual work for syscopy).

rmscopy($from,$to[,$date_flag])

The first and second arguments may be strings, typeglobs, typeglob references, or objects inheriting from IO::Handle; they are used in all cases to obtain the *filespec* of the input and output files, respectively. The name and type of the input file are used as defaults for the output file, if necessary.

A new version of the output file is always created, which inherits the structure and RMS attributes of the input file, except for owner and protections (and possibly timestamps; see below). All data from the input file is copied to the output file; if either of the first two parameters to `rmscopy` is a file handle, its position is unchanged. (Note that this means a file handle pointing to the output file will be associated with an old version of that file after `rmscopy` returns, not the newly created version.)

The third parameter is an integer flag, which tells `rmscopy` how to handle timestamps. If it is < 0, none of the input file's timestamps are propagated to the output file. If it is > 0, then it is interpreted as a bitmask: if bit 0 (the LSB) is set, then timestamps other than the revision date are propagated; if bit 1 is set, the revision date is propagated. If the third parameter to `rmscopy` is 0, then it behaves much like the DCL COPY command: if the name or type of the output file was explicitly specified, then no timestamps are propagated, but if they were taken implicitly from the input filespec, then all timestamps other than the revision date are propagated. If this parameter is not supplied, it defaults to 0.

Like `copy`, `rmscopy` returns 1 on success. If an error occurs, it sets `$!`, deletes the output file, and returns 0.

## RETURN

All functions return 1 on success, 0 on failure. $! will be set if an error was encountered.

## NOTES

- On Mac OS (Classic), the path separator is ':', not '/', and the current directory is denoted as ':', not '.'. You should be careful about specifying relative pathnames. While a full path always begins with a volume name, a relative pathname should always begin with a ':'. If specifying a volume name only, a trailing ':' is required.

  E.g.

```
  copy("file1", "tmp");        # creates the file 'tmp' in the
current directory
```

```
  copy("file1", ":tmp:");      # creates :tmp:file1
  copy("file1", ":tmp");       # same as above
  copy("file1", "tmp");        # same as above, if 'tmp' is a
directory (but don't do
                               # that, since it may cause confusion,
see example #1)
  copy("file1", "tmp:file1");  # error, since 'tmp:' is not a volume
  copy("file1", ":tmp:file1"); # ok, partial path
  copy("file1", "DataHD:");    # creates DataHD:file1

  move("MacintoshHD:fileA", "DataHD:fileB"); # moves (doesn't copy)
files from one
                                             # volume to another
```

## AUTHOR

File::Copy was written by Aaron Sherman <*ajs@ajs.com*> in 1995, and updated by Charles Bailey <*bailey@newman.upenn.edu*> in 1996.