

# How to Easily Move Files Between Folders in R

Authored by  
**stats writer**

November 19, 2025

## RECOMMENDED CITATION

stats writer (2025). *How to Easily Move Files Between Folders in R*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=97291>

Effective R programming often requires precise control over the filesystem, including the ability to manipulate, organize, and relocate data files. Moving files is a fundamental operation necessary for managing project workflows, archiving results, or setting up reproducible data pipelines. The primary function recommended for achieving a true move--relocation followed by deletion of the original--is the `file.rename()` function. This powerful utility takes two mandatory arguments: the full `file_path` of the source file and the desired file path for the destination. By updating the file's directory location metadata, it effectively moves the file, deleting the source instance.

The source and destination locations must be meticulously specified. The source file must exist and be accessible, and the destination must be a writable directory. The function is designed to handle local directory paths; while URLs can sometimes be interacted with using other specialized packages, standard file operations rely on local or networked filesystem paths. Crucially, the outcome of the operation is communicated via a `Boolean value`, ensuring that scripts can verify success (`TRUE`) or failure (`FALSE`) before proceeding to dependent steps.

You can use two distinct methods to handle file relocation in R, depending on whether the original file must be preserved or deleted:

## Introduction to File Management in R

Managing files directly within the R environment provides powerful capabilities, eliminating the need to manually interact with the operating system's file explorer. This seamless integration ensures that data handling processes are scriptable, repeatable, and easily scalable across different projects or collaborators. A key challenge in file management is ensuring that absolute or relative paths are correctly specified, particularly when working across different operating systems like Windows, macOS, and Linux, which use varying conventions for directory separators.

When we discuss moving a file in a computational context, we are generally referring to changing its directory location. In R, this is accomplished using functions that either create a new copy of the file in the destination and then delete the source (the standard move behavior achieved by `file.rename()`), or functions that only perform the duplication (`file.copy()`). The appropriate function depends entirely on the specific requirements of your workflow--whether duplication is acceptable or if immediate removal of the original file is necessary to save space or prevent confusion.

We will examine the two primary functions used for file transfer in R: `file.copy()`, which duplicates the file, and `file.rename()`, which performs a direct move operation by deleting the source file. Both functions require careful specification of the `from` (source) and `to` (destination) arguments, which must include the full file names, not just the directory names, to execute the relocation successfully.

## Choosing the Right R Function: Copy vs. Rename

The distinction between copying and renaming a file path is fundamental to efficient file manipulation in R. While a manual move operation often combines both steps (copy followed by deletion), R explicitly separates these actions into distinct functions, giving the user granular control over the process. Choosing between `file.copy()` and `file.rename()` hinges on one simple question: Do you require the original file to remain intact in its source location after the operation completes?

If the goal is to create a backup or place the same dataset into multiple processing folders without altering the source data, then the `file.copy()` function is appropriate. This operation is safer for non-destructive workflows but consumes more disk space as two identical files coexist temporarily or permanently. Conversely, if the objective is to truly move the file--transferring ownership of the file to the new directory while simultaneously cleaning up the old location--then `file.rename()` is the correct tool for performing the atomic relocation.

It is important to remember that both functions are sensitive to permissions. If R does not have the necessary writing permissions for the destination directory, or reading permissions for the source file, the operation will fail silently and return `FALSE`. Therefore, prior verification of directory existence and access rights is a crucial step before attempting file relocation, particularly when dealing with critical system directories or shared network drives.

### Method 1: Duplicating Files Using `file.copy()` (Keep Original File)

When the requirement is to transport a file to a new location while ensuring the integrity and persistence of the original data in its current directory, the `file.copy()` function is the definitive solution. This method is often employed for making incremental backups, preparing data subsets for external analysis, or ensuring the source dataset remains untouched during iterative processing phases.

```
file.copy(from="C:/Users/bob/Documents/current_data/soccer_data.csv",  
to="C:/Users/bob/Documents/new_data/soccer_data.csv")
```

This particular syntax copies a file called `soccer_data.csv` located in the following source directory, specified by its full file path:

**C:/Users/bob/Documents/current\_data**

The identical file is then duplicated and saved into the following destination folder:

**C:/Users/bob/Documents/new\_data**

Since we used the `file.copy()` function, the **original file is still kept** in the folder it came from, resulting in two identical copies residing on the filesystem.

## Method 2: Moving Files by Renaming Using `file.rename()` (Delete Original File)

When the objective is to execute a genuine move operation--transferring the file to a new location and deleting the source--the `file.rename()` function is the required tool. This function is inherently more destructive than `file.copy()` because the original source file is removed upon successful completion. It should be used for cleanup or archival processes where retaining the original file is unnecessary or undesirable.

```
file.rename(from="C:/Users/bob/Documents/current_data/soccer_data.csv",  
to="C:/Users/bob/Documents/new_data/soccer_data.csv")
```

This particular syntax instructs **R** to relocate the file called **soccer\_data.csv** currently situated in the following source folder:

**C:/Users/bob/Documents/current\_data**

The file is then transferred into the following destination folder:

**C:/Users/bob/Documents/new\_data**

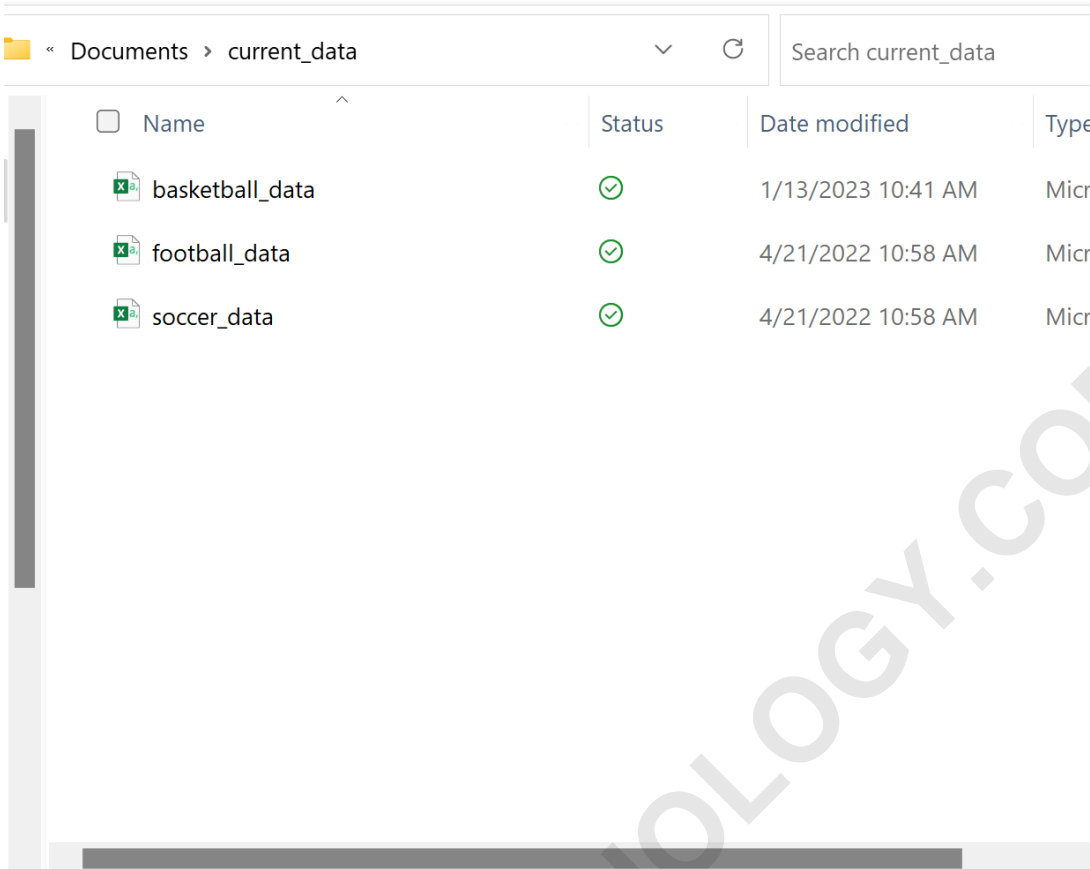
Since we used the `file.rename()` function, the original file is deleted from the folder it came from, completing the destructive move operation. The following examples show how to use each method in practice, including visual confirmation of the results.

## Example 1: Move File in R (And Keep Original File)

### Visualizing the Copy Operation

Suppose we want to duplicate a file called **soccer\_data.csv** located in the source folder: **C:/Users/bob/Documents/current\_data**. This folder currently holds several files, representing the initial state of our data structure.

The folder currently has three files in it, as displayed below:



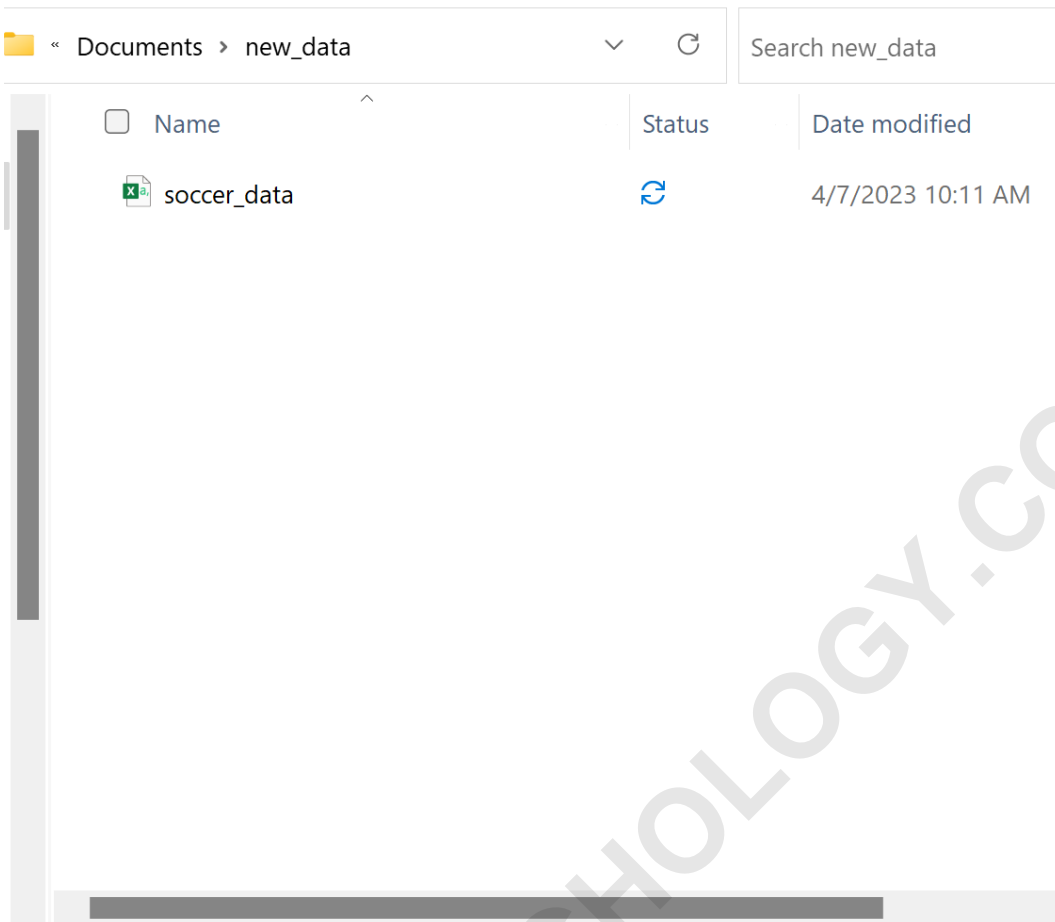
We use the following syntax in R, leveraging `file.copy()`, to move **soccer\_data.csv** into a new folder called **new\_data**, ensuring the source file remains untouched:

```
file.copy(from="C:/Users/bob/Documents/current_data/soccer_data.csv",  
to="C:/Users/bob/Documents/new_data/soccer_data.csv")
```

```
TRUE
```

The `file.copy()` function returns a value of **TRUE**, which tells us that the duplication process worked successfully. This immediate feedback is invaluable for scripting automated processes, confirming that the new copy has been created and written to disk.

If we open the **new\_data** folder, we can see that the **soccer\_data.csv** file has indeed been copied into it, confirming the successful creation of the duplicate file:



And if we return to the source folder where the file came from, we will see that the original file still exists there, validating the non-destructive nature of the `file.copy()` function.

**Note:** If either of the `file_path` specifications is incorrect, or if R lacks the necessary write permissions in the target directory, the `file.copy()` function will instead return a Boolean value of **FALSE**, indicating failure.

## Example 2: Move File in R (And Delete Original File)

### Executing the True Move Operation

In this scenario, we perform a true move, utilizing the `file.rename()` function to transfer the file **soccer\_data.csv** to the **new\_data** folder while simultaneously deleting the instance from the source directory, **current\_data**. This action is irreversible without backups, so precision is paramount.

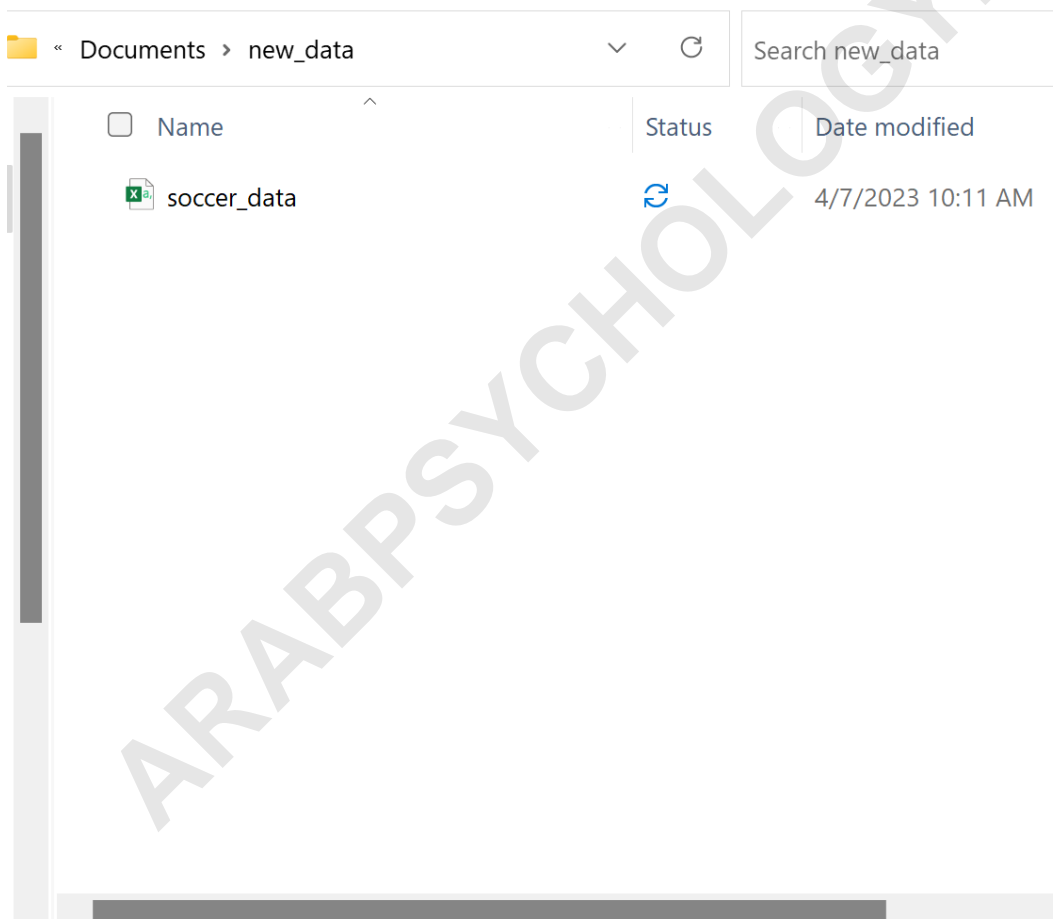
We use the following syntax in R to move **soccer\_data.csv** into a new folder called **new\_data** and delete the original file:

```
file.rename(from="C:/Users/bob/Documents/current_data/soccer_data.csv",  
to="C:/Users/bob/Documents/new_data/soccer_data.csv")
```

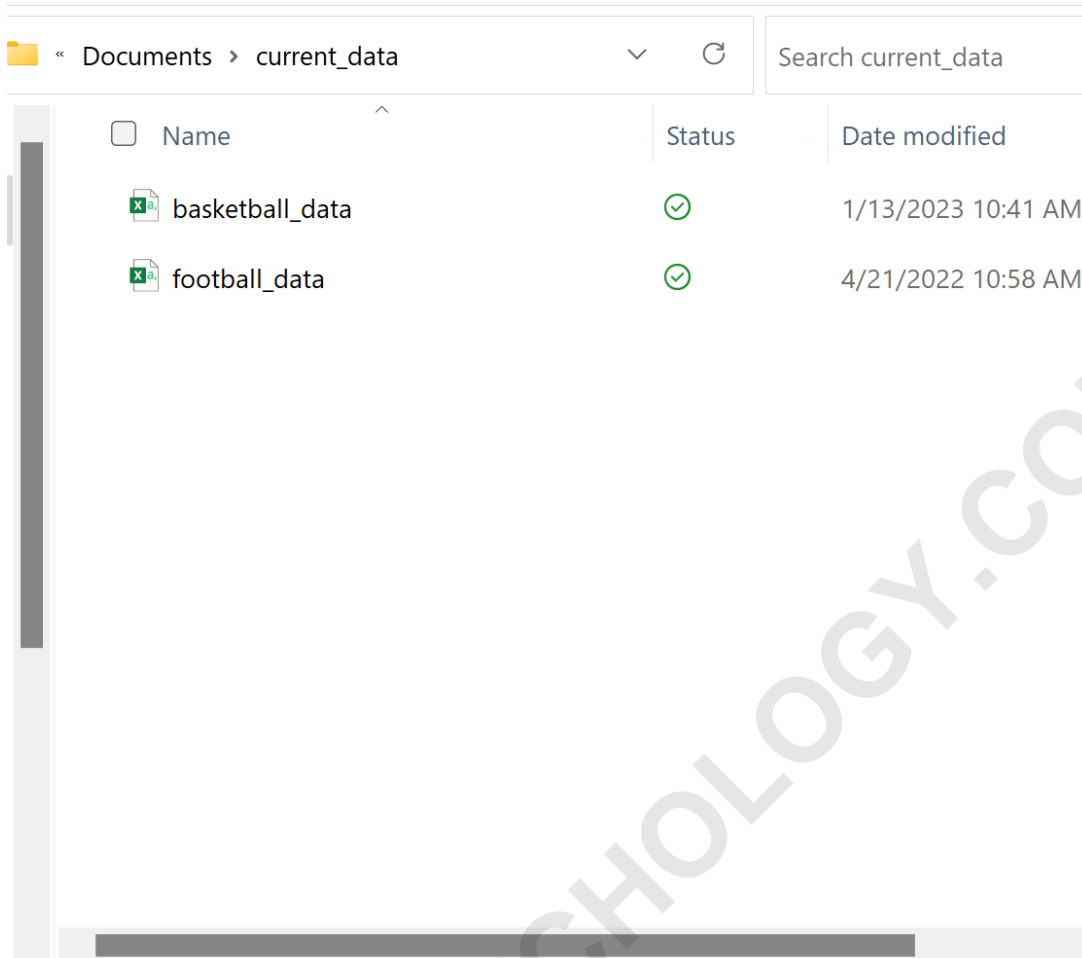
```
TRUE
```

The `file.rename()` function returns a value of **TRUE**, which confirms that the move operation was completed successfully. This function acts as an atomic operation, meaning the file is either successfully moved and the source deleted, or the operation fails entirely, leaving the source file untouched.

If we open the **new\_data** folder, we can see that the **soccer\_data.csv** file has indeed been relocated into it:



The crucial difference is demonstrated when we return to the source folder, **current\_data**. We observe that **soccer\_data.csv** has been successfully deleted from that folder, leaving only the two remaining files:



## Troubleshooting and Best Practices for File Paths

Working with file paths across different operating systems or environments presents one of the most common challenges in R file management. The R language is highly flexible but requires precise path specification. Using the correct format is paramount for avoiding the dreaded `FALSE` return value that indicates a failed operation, whether using `file.copy()` or `file.rename()`.

There are several best practices data scientists should adopt when scripting file movements in R:

**Use Forward Slashes:** Even when working on Windows, R interprets forward slashes (/) reliably as directory separators. It is the preferred, cross-platform standard in R programming, whereas Windows backslashes (\) often require tedious escaping (\\).

**Verify Existence:** Always ensure that the destination directory exists before attempting a move or copy operation. Functions like `dir.exists()` and `dir.create()` help manage target folders dynamically, preventing errors caused by missing directories.

**Check Permissions:** Ensure R has both read access to the source file and write access to the destination directory. Lack of permissions is a frequent cause of unexpected failure, particularly in

managed network environments.

Finally, always confirm the return value of the operation. The immediate feedback of a Boolean value (`TRUE` or `FALSE`) is your primary indicator of success. If a move or copy operation returns `FALSE`, the issue is highly likely related to an incorrect path specification, file non-existence, or permission limitations within the operating system.

ARABPSYCHOLOGY.COM