# How to Easily Rename Files in R Using file.rename()

Authored by

**stats writer**

December 2, 2025

Effective data management is crucial for reproducible research and streamlined workflow, especially when utilizing the R programming language for statistical computing and graphics. One common administrative task is renaming data files or reports after processing them. While many users might resort to manual operating system commands, the built-in `file.rename()` function provides a robust, platform-independent solution directly within your R scripts.

The core utility for altering file names in R is `file.rename()`. This function is designed to facilitate quick and efficient organization of your data assets. It operates by requiring two primary arguments: the path and name of the file to be renamed (the `from` argument) and the desired new path and name (the `to` argument). Successful execution of this command is predicated on R's ability to correctly locate the file based on the specified working directory or the explicit file path provided.

Learning how to utilize this function effectively is vital for maintaining an organized project structure. Whether you are correcting a typo in a filename, archiving old versions, or standardizing naming conventions across a large dataset of CSV files, integrating file renaming into your R workflow ensures that your processes remain automated and easy to trace. This guide will explore two powerful methods for leveraging this capability, ranging from renaming a single file to complex, pattern-based batch operations.

## Fundamental Methods for File Renaming in R

When approaching file management tasks in R, it is helpful to categorize the operations based on the scale of transformation required. The two most common scenarios involve either altering the name of a singular, specific file or conducting a mass renaming operation across numerous files that share a common naming characteristic or pattern. R provides tailored, efficient solutions for both situations, ensuring maximal control and minimal manual intervention.

Below are the foundational methods we will explore in detail. Understanding the distinction between these approaches--the precise, one-to-one mapping versus the complex pattern substitution--is key to choosing the correct function parameters and external library dependencies, such as the powerful stringr package for vectorized text manipulation. We will begin by establishing the syntax for simple renaming, which relies solely on base R functionality, before progressing to advanced techniques necessary for handling large repositories of data where systematic changes are required.

You can use the following methods to rename files in R:

**Method 1: Rename One File**

**file.rename(from='old_name.csv', to='new_name.csv')**

**Method 2: Replace Pattern in Multiple Files**

**file.rename(list.files(pattern ='old'),**
**str_replace(list.files(pattern='old'), pattern='old', 'new'))**

The following examples show how to use each method in practice.

## Method 1: Renaming a Single File

The simplest application of the file renaming utility involves changing the name of one specific file. This method is straightforward and relies exclusively on the base R function, `file.rename()`. To successfully execute this operation, you must ensure that R has read/write permissions for the file and that the file path specified in the `from` argument is completely accurate. Any minor discrepancy in the path or filename will result in a failure, often returning a logical vector of `FALSE` without an explicit warning detailing the exact location of the error.

The syntax requires two essential arguments. The first, `from`, must contain a character string specifying the current, complete name of the file you intend to modify. The second, `to`, must contain a character string representing the desired new name. It is important to note that if you are working within the current working directory, you only need to provide the filename itself. If the file resides elsewhere, you must provide the full, relative, or absolute path for both the `from` and `to` arguments. Crucially, the `to` argument essentially determines the file's new location; if you provide a path different from the `from` path, this operation acts as both a rename and a move.

A key consideration when using `file.rename()` for single files is error handling. If the source file specified in `from` does not exist, or if a file with the name specified in `to` already exists and R cannot overwrite it due to permissions or settings, the function will fail silently or return an error indicator. Therefore, always verify the existence of the source file beforehand and ensure that the target name is either unique or safely overwritable, depending on your intended outcome. This proactive approach minimizes interruptions in larger data processing pipelines.

## Code Example: Rename One File

To illustrate the single file renaming process, let us assume we have established a working directory containing several data files that require standardization. We can first inspect the contents of this directory using the `list.files()` function to confirm the current state of our files. Suppose we have four CSV files, and one of them, `data1.csv`, needs to be updated to follow the consistent naming convention established by the others.

**#display all files in current working directory**

**list.files()**

"data1.csv" "data2_good.csv" "data3_good.csv" "data4_good.csv"

We can use the following code, invoking `file.rename()`, to rename the file called **data1.csv** to **data1_good.csv**. Notice the direct mapping of the old name to the new name within the function's arguments. This operation is atomic and immediate upon execution, assuming permissions are correctly configured.

**#rename one file**
**file.rename(from='data1.csv', to='data1_good.csv')**

#display all files in current working directory
list.files()

"data1_good.csv" "data2_good.csv" "data3_good.csv" "data4_good.csv"

Upon reviewing the output of `list.files()` after the command executes, we clearly observe that the file has been successfully renamed. This confirms the successful execution of the base R function for singular file operations, providing a quick check that the desired outcome has been achieved without manual verification in the file system.

## Method 2: Batch Renaming Files Using Pattern Replacement

While renaming a single file is simple, the true power of R's file management capabilities emerges when dealing with batch operations. Renaming multiple files simultaneously requires a structured approach that typically involves three distinct steps: identifying the target files, defining the renaming transformation logic, and applying the `file.rename()` function to the resulting vectors of old and new names. This technique is indispensable for large projects where consistent restructuring of file names is essential.

To identify the target files, we rely on the base R function `list.files()`, often combined with its `pattern` argument. This argument accepts regular expressions, allowing the user to filter the contents of the directory based on specific text sequences within the filenames. For instance, if you only want to affect files containing the substring "good," you pass that string to the `pattern` argument, yielding a character vector containing only the names of the selected files.

The second, and arguably most crucial, step is defining the transformation logic. Since standard base R functions can be cumbersome for sophisticated string manipulation, we often utilize functions from specialized packages like stringr package (part of the Tidyverse). The `str_replace()` function from stringr is ideal for this task, as it efficiently takes a vector of

filenames and replaces a specified pattern (e.g., "good") with a new string (e.g., "bad") across all elements in the vector. This results in two parallel vectors of equal length: one containing the original names and one containing the new, transformed names.

Finally, these two vectors--the output of `list.files()` (the `from` argument) and the output of `str_replace()` (the `to` argument)--are passed simultaneously to the `file.rename()` function. R then iterates through both vectors, renaming each file element by element. This vectorized operation significantly improves performance and script readability compared to writing a traditional loop, making it the preferred method for high-volume file management.

## Code Example: Replace Pattern in Multiple Files

Let's continue with the scenario where all our files now contain the suffix "_good," but due to a change in project specifications, we need to replace that pattern with "_bad" in every filename. This is a classic use case for vectorized pattern replacement. First, we confirm the current state of our directory using `list.files()`.

**#display all files in current working directory**
**list.files()**

"data1_good.csv" "data2_good.csv" "data3_good.csv" "data4_good.csv"

To execute the batch rename, we must first load the necessary library, `stringr`, which provides the robust `str_replace` functionality. The first argument to `file.rename()` is the vector of current names, generated by filtering files that contain "good." The second argument generates the new names by applying `str_replace()` to that same list of files, swapping "good" for "bad."

**library(stringr)**

file.rename(list.files(pattern ='good'),
str_replace(list.files(pattern='good'), pattern='good', 'bad'))

#display all files in current working directory
list.files()

"data1_bad.csv" "data2_bad.csv" "data3_bad.csv" "data4_bad.csv"

The final output confirms that all files matching the criteria were successfully renamed, demonstrating the efficiency and declarative nature of R's vectorized file operations. This method ensures consistency across large datasets and minimizes the risk of human error associated with manual renaming.

## Best Practices and Error Handling

While `file.rename()` is powerful, several best practices should be observed to ensure data integrity and script reliability. Firstly, always perform a dry run before executing any mass renaming operation. A dry run involves generating the vector of new names using `str_replace()` and printing both the original (`from`) and new (`to`) vectors side-by-side without actually calling `file.rename()`. This allows you to visually inspect the transformations and catch any unintended consequences of your pattern matching before files are permanently altered.

Secondly, pay close attention to the definition of your working directory. Relative paths are dependent on the current working directory (CWD), which can change depending on how your script is run (e.g., interactively versus batch processing). For maximum reliability, especially in package development or large projects, it is recommended to use the `here` package or explicitly define absolute paths, or at least manage paths relative to the project root.

When R fails to rename a file, `file.rename()` returns a logical value of `FALSE` for that specific operation. Common causes for failure include permission denied errors (often encountered when trying to rename files created by another user or files currently open in another program), or non-existent files. If the `from` file path is incorrect, R returns `FALSE`. If the `to` file path already exists and R cannot overwrite it, it also returns `FALSE`. To troubleshoot, you can wrap the `file.rename()` call within a structure that checks permissions or file existence first. For instance, using `file.exists()` can preemptively confirm that the file you intend to rename is actually present.

## Conclusion: Streamlining Your R Workflow

Mastering file system interactions within the R programming language is a cornerstone of efficient and reproducible data analysis. By utilizing the `file.rename()` function, either for specific singular changes or for complex batch pattern replacements coordinated with `list.files()` and external packages like stringr, researchers and developers can automate laborious administrative tasks. This automation not only saves time but also enforces consistency, which is vital when working with dynamic datasets.

The ability to programmatically manage filenames ensures that subsequent analytical scripts--which often rely on specific naming conventions for input data--will execute without manual intervention. This moves the user away from manual file system adjustments toward fully automated, self-contained R scripts that are easily shareable and scalable across different project stages and collaborators.

Remember that robust file management is not just about renaming; it is about establishing a foundational structure for your entire data pipeline. Always prioritize clarity, utilize dry runs, and confirm successful execution using the logical return values provided by the R functions. By

integrating these practices, file renaming transitions from a chore into a reliable and integral part of your professional R programming language workflow.

## Further Resources for File Management

The following tutorials explain how to perform other common operations with files in R: