How to List Files by Date in R (With Example)

Authored by

stats writer

November 19, 2025

Effective data management in R often requires users to interact directly with the file system, especially when dealing with large projects or dynamic data sources. One fundamental task is efficiently listing files and directories, particularly when sorting them based on creation, modification, or access dates. The primary tool for navigating the file system in R is the `list.files()` function, which serves as the foundational step in this process.

While `list.files()` by itself returns a simple vector of filenames, it does not inherently provide the timestamp information needed for sorting. To achieve date-based sorting, we must combine `list.files()` with `file.info()`, which retrieves detailed metadata about the files. This combination allows R users to treat file system information like any other data frame, enabling powerful filtering, selection, and, crucially, sorting operations.

The `list.files()` function is highly flexible, supporting various arguments that refine the output. Key arguments include `pattern`, which restricts the output to files matching a specific regular expression (e.g., all CSV files), and `recursive`, which determines if the listing should traverse subdirectories. Additionally, setting `all.files=TRUE` ensures that hidden files are included in the result, and `full.names=TRUE` is often essential for returning the complete file path, preventing ambiguity when dealing with files across different directories. Mastering these parameters is vital for robust file management workflows.

Understanding the Core Tools: list.files() and file.info()

To successfully list files sorted by date, two core R functions must work in tandem. First, `list.files()` identifies the files we are interested in. When this function is called, it returns a character vector containing the names of the files and folders residing in the specified directory, or the current working directory by default.

Second, `file.info()` is then applied to the output of `list.files()`. Unlike its counterpart, `file.info()` does not return filenames but rather a data frame containing comprehensive metadata for each file passed to it. This metadata includes columns such as `size` (file size), `isdir` (whether it's a directory), `mode` (permissions), and, most importantly for our purposes, the three critical timestamp columns: `mtime`, `ctime`, and `atime`.

By chaining these two functions--passing the results of `list.files()` directly into `file.info()`--we create a structured data frame that contains both the file identifier (as row names) and all the necessary date information. This resulting structure forms the basis upon which we can perform sophisticated sorting and filtering operations necessary for organizing files according to temporal sequence, ensuring that data pipelines are executed in the correct chronological order.

The Role of Timestamps: Mtime, Ctime, and Atime

When working with file metadata, it is critical to understand the specific meaning of the three primary timestamps provided by the operating system and exposed by R via `file.info()`. These timestamps represent different events in a file's lifecycle and are crucial for determining the correct sorting criteria for any data management task.

The `mtime`, or modification time, tracks the last moment the file's content was written or changed. This is the most common timestamp used when attempting to sort files by when the data they contain was last updated. If you are looking for the newest version of a script or the most recently processed dataset, `mtime` is the column you should utilize for sorting.

In contrast, `ctime` represents the status change time. On Unix-like systems, `ctime` updates when file attributes are changed, such as permissions, ownership, or the number of hard links, in addition to when the file content is modified. While often close to the modification time, it serves a slightly different technical purpose regarding file system integrity. Finally, `atime` records the last access time, documenting when the file was last read or executed. This timestamp can be less reliable for general sorting purposes as many modern operating systems or caching mechanisms suppress or delay `atime` updates for performance reasons.

Understanding the difference between these three timestamps--mtime, ctime, and atime--is essential for accurate data handling. Choosing the correct timestamp ensures that the resulting sorted list meets the specific requirements of the analytical task, whether it involves finding the latest version of a file or tracking its reading history.

Syntax Breakdown: Sorting Files by Modification Date (mtime)

Once the file metadata is gathered into a data frame using `file.info(list.files(...))`, the next step involves applying a sort mechanism to order the files based on a chosen timestamp column, such as `mtime`. This sorting is achieved using R's powerful indexing and ordering functions.

The primary sorting tool is the `order()` function. This function does not sort the data frame itself; instead, it returns a vector of indices that, if applied to the data frame, would arrange its rows in the desired order. When sorting based on dates, it is crucial to ensure that the timestamp column is correctly interpreted as a date-time object rather than a character string.

Since the timestamp columns (`mtime`, `ctime`, `atime`) returned by `file.info()` are already in the `POSIXct` format, which represents calendar dates and times, they are ready for numerical sorting. The sorting expression typically utilizes the `order()` function within the indexing brackets of the data frame. We use the `with()` function to simplify the syntax, allowing direct reference to the

column names within the sorting command. The final result is a new data frame where rows (files) are arranged sequentially from oldest to newest based on the modification timestamp.

The basic sequence for sorting files by date in the current working directory involves filtering, extracting metadata, and applying the `order()` function:

```
#extract all CSV files in working directory
file_info = file.info(list.files(pattern="*.csv"))

#sort files based on mtime (modification date and time)
file_info = file_info

#view only file names with modification date and time
file_info
```

The following practical scenario demonstrates how to apply this syntax in a real-world file management context.

Practical Example: Listing Specific File Types (CSV)

Suppose a user is working in an R environment that contains numerous data files of various types, but the task requires specifically listing and sorting only the CSV files in the current working directory based on their last modification date. This scenario is common in data science workflows where historical tracking and chronological processing are necessary.

We begin by using the `list.files()` function, incorporating the `pattern` argument set to `"*.csv"`. This crucial step filters the output immediately, restricting the results to only the desired file type. The output of this filtered list is then piped into `file.info()`, which generates a comprehensive data frame containing all the necessary metadata, including the modification timestamps for every relevant file.

The following syntax extracts the metadata for all CSV files in the current working directory, storing the result in the `file_info` object:

```
#extract all CSV files in working directory
file_info = file.info(list.files(pattern="*.csv"))

#view all CSV files and their metadata
file_info
```

size isdir mode mtime ctime atime exe

basketball_data.csv 55 FALSE 666 2023-01-06 11:07:43 2022-07-12 09:07:26 2023-04-18 09:42:19 no
df1.csv 126 FALSE 666 2022-04-21 10:48:24 2022-04-21 10:48:24 2023-04-18 09:42:19 no
df2.csv 126 FALSE 666 2022-04-21 10:48:30 2022-04-21 10:48:29 2023-04-18 09:42:19 no
df3.csv 126 FALSE 666 2022-04-21 10:48:34 2022-04-21 10:48:34 2023-04-18 09:42:19 no
my_data.csv 53 FALSE 666 2022-09-09 09:02:21 2022-04-22 09:00:13 2023-04-18 09:42:19 no
my_list.csv 90 FALSE 666 2022-04-21 09:40:01 2022-04-21 09:39:59 2023-04-18 09:42:19 no
my_test.csv 146 FALSE 666 2022-04-21 09:42:25 2022-04-21 09:42:25 2023-04-18 09:42:19 no
player_stats.csv 137 FALSE 666 2023-04-11 09:07:20 2023-04-11 09:07:20 2023-04-18 09:42:19 no
players_data.csv 50 FALSE 666 2023-01-06 09:44:12 2023-01-06 09:44:12 2023-04-18 09:42:19 no
team_info.csv 131 FALSE 666 2023-04-11 09:07:21 2023-04-11 09:07:21 2023-04-18 09:42:19 no
test.csv 18059168 FALSE 666 2022-09-07 09:07:34 2020-02-01 13:44:03 2023-04-18 09:42:19 no
uneven_data.csv 43 FALSE 666 2023-01-06 14:02:17 2023-01-06 14:00:27 2023-04-18 09:42:19 no

Note that the output shows the file names as row names and provides multiple columns of data. Crucially, the `mtime` column confirms the exact date and time of the last modification for each file, which is currently unsorted. The next step is to apply the chronological ordering based on this column.

Applying Sorting Based on Modification Time (mtime)

To properly order the files chronologically, we utilize the `order()` function, instructing R to sort the `file_info` data frame based specifically on the `mtime` column. Since the `mtime` column is already in the appropriate POSIXct date-time format, the sorting operation handles the temporal sequence correctly, arranging files from the oldest modification date to the newest.

The sorting operation is accomplished by passing the `mtime` column to `order()` and then using the resulting indices to reorder the rows of the `file_info` data frame. This is a standard and highly efficient method in R for arranging data frames based on the values of a specific column.

The following code executes the sort and displays the reorganized file information, which is now sorted chronologically based on when the files were last changed:

#sort files based on mtime (modification date and time)
file_info = file_info

#view sorted files

file_info

size isdir mode mtime ctime atime exe
my_list.csv 90 FALSE 666 2022-04-21 09:40:01 2022-04-21 09:39:59 2023-04-18 09:42:19 no
my_test.csv 146 FALSE 666 2022-04-21 09:42:25 2022-04-21 09:42:25 2023-04-18 09:42:19 no
df1.csv 126 FALSE 666 2022-04-21 10:48:24 2022-04-21 10:48:24 2023-04-18 09:42:19 no
df2.csv 126 FALSE 666 2022-04-21 10:48:30 2022-04-21 10:48:29 2023-04-18 09:42:19 no
df3.csv 126 FALSE 666 2022-04-21 10:48:34 2022-04-21 10:48:34 2023-04-18 09:42:19 no
test.csv 18059168 FALSE 666 2022-09-07 09:07:34 2020-02-01 13:44:03 2023-04-18 09:42:19 no
my_data.csv 53 FALSE 666 2022-09-09 09:02:21 2022-04-22 09:00:13 2023-04-18 09:42:19 no
players_data.csv 50 FALSE 666 2023-01-06 09:44:12 2023-01-06 09:44:12 2023-04-18 09:42:19 no
basketball_data.csv 55 FALSE 666 2023-01-06 11:07:43 2022-07-12 09:07:26 2023-04-18 09:42:19 no
uneven_data.csv 43 FALSE 666 2023-01-06 14:02:17 2023-01-06 14:00:27 2023-04-18 09:42:19 no
player_stats.csv 137 FALSE 666 2023-04-11 09:07:20 2023-04-11 09:07:20 2023-04-18 09:42:19 no
team_info.csv 131 FALSE 666 2023-04-11 09:07:21 2023-04-11 09:07:21 2023-04-18 09:42:19 no

It is important to remember that this process is easily adaptable. If the requirement was to order files based on creation date, one would simply substitute `ctime` for mtime within the `order()` function. Similarly, using `atime` would sort the files based on the last time they were accessed.

Refining the Output: Displaying Specific Date and Time Columns

While the full `file_info` data frame provides extensive details, data analysis tasks often require only a concise view of the results--specifically, the file names and their corresponding modification dates. After the sorting operation is complete, the final step involves subsetting the data frame to present only the necessary columns.

By using standard R bracket notation (), we can select only the `mtime` column from the now-sorted `file_info` data frame. Since the file names are preserved as row names in the resulting output, this provides a clean, chronological list that directly answers the need for date-based file organization.

We subset the data frame using the column name `"mtime"` to isolate the file names alongside the date and time they were most recently modified:

#view only file names with modification date and time

file_info

mtime
my_list.csv 2022-04-21 09:40:01
my_test.csv 2022-04-21 09:42:25
df1.csv 2022-04-21 10:48:24
df2.csv 2022-04-21 10:48:30
df3.csv 2022-04-21 10:48:34
test.csv 2022-09-07 09:07:34
my_data.csv 2022-09-09 09:02:21
players_data.csv 2023-01-06 09:44:12
basketball_data.csv 2023-01-06 11:07:43
uneven_data.csv 2023-01-06 14:02:17
player_stats.csv 2023-04-11 09:07:20
team_info.csv 2023-04-11 09:07:21

For scenarios where only the chronological sequence of filenames is required, the `rownames()` function can be applied to the sorted `file_info` object. This extracts the row labels (which correspond to the file names) in the sequence dictated by the preceding date-based sort. This provides the most simplified output for subsequent automation or processing.

#view only file names
rownames(file_info)

"my_list.csv" "my_test.csv" "df1.csv" "df2.csv" "df3.csv"
"test.csv" "my_data.csv" "players_data.csv" "basketball_data.csv" "uneven_data.csv"
"player_stats.csv" "team_info.csv"

As shown in the output above, the twelve CSV file names are now listed precisely in order based on their modification date and time, starting with the oldest file and concluding with the newest.

Conclusion: Leveraging Date Sorting for Data Workflow

The ability to list and sort files chronologically in R is a fundamental skill for maintaining clean, reproducible, and efficient data workflows. By strategically combining `list.files()` and `file.info()`, users gain access to critical file system metadata that allows for temporal organization.

Whether you need to automatically process data files in the order they were generated, archive outdated versions, or identify the most recent iteration of an analytical result, sorting by timestamp

provides the necessary structure. This method ensures accuracy and eliminates manual errors associated with relying on arbitrary file name conventions or directory order.

Ultimately, mastering the use of functions like `order()` in conjunction with timestamp fields like mtime allows R users to integrate operating system details directly into their data analysis pipeline, enhancing the overall robustness and automation capabilities of their scripts.