How to Extract the Last Item After Splitting Text in Excel

Authored by

stats writer

February 12, 2026

Excel: Split Text and Get Last Item

In the modern landscape of data management, **Microsoft Excel** remains an indispensable tool for professionals across various industries. One of the most frequent challenges users face involves the manipulation of **string** data, particularly when information is concatenated within a single cell. Whether you are dealing with full names, complex inventory codes, or file paths, the ability to isolate specific components from a text string is a fundamental skill. Historically, this required convoluted nested formulas involving functions like **FIND**, **LEN**, and **MID**. However, the introduction of **dynamic array** functions has revolutionized this process, making it more intuitive and efficient than ever before.

The **TEXTSPLIT** function in **Microsoft Excel** is a powerful utility designed to parse text based on a specific **delimiter**. By identifying a character--such as a space, comma, or semicolon--this function can automatically distribute segments of text into separate cells. This functionality is particularly useful when you need to deconstruct data that has been exported from other systems or databases where fields are merged into a single column. Understanding how to leverage this function is the first step toward mastering complex text manipulation tasks within your spreadsheets.

To retrieve the last item that results from this function, you can use the following syntax:

=CHOOSECOLS(TEXTSPLIT(A2, " "), -1)

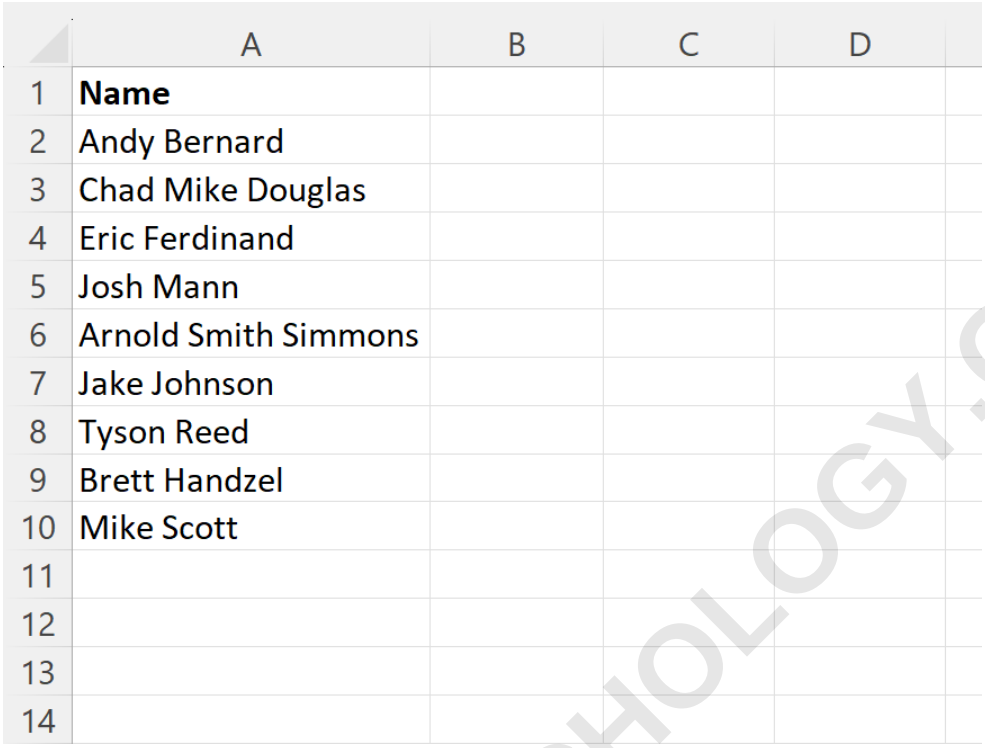
This particular example will split the text in cell **A2** using a space as a **delimiter** and then it will return only the last item that results from the split. By combining the **TEXTSPLIT** function with the **CHOOSECOLS** function, users can create a highly efficient workflow for data extraction. The use of the **-1** argument within the **CHOOSECOLS** function is a critical component of this formula, as it instructs **Microsoft Excel** to count backward from the end of the **array**, ensuring that the final element is always selected regardless of how many segments the text is split into.

For example, if the string in cell **A2** is **Chad Mike Douglas** then this formula will return **Douglas**. In this scenario, the name is divided into three distinct parts: "Chad", "Mike", and "Douglas". Without a dynamic approach, identifying the last name could be difficult if some entries in your list contain middle names while others do not. This formula provides a robust solution that adapts to the varying lengths of input strings, ensuring consistent results across your entire dataset without the need for manual adjustments or complex logical statements.

The following example shows how to use this formula in practice.

Example: How to Split Text and Get Last Item in Excel

Suppose we have the following column of names in **Microsoft Excel**:



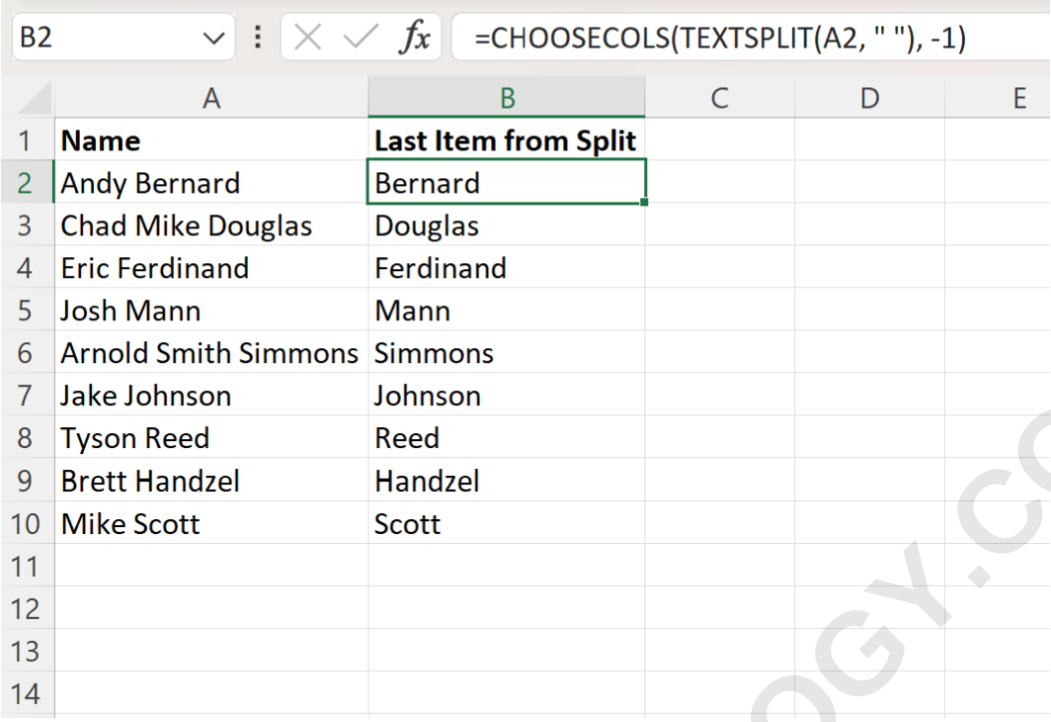
	A	B	C	D
1	**Name**			
2	Andy Bernard			
3	Chad Mike Douglas			
4	Eric Ferdinand			
5	Josh Mann			
6	Arnold Smith Simmons			
7	Jake Johnson			
8	Tyson Reed			
9	Brett Handzel			
10	Mike Scott			
11				
12				
13				
14				

In many administrative and data analysis tasks, you may encounter datasets where full names are provided in a single field. Suppose we would like to split the names based on where a space occurs and then get only the last item that results from the split. This is a common requirement when you need to sort a list by surname or when preparing a mailing list where only the last name is required for formal addressing. The traditional method of using **Text to Columns** is permanent and non-dynamic, meaning any changes to the source data would require you to repeat the process. Using a formula ensures that your output remains synchronized with your source data.

To implement this dynamically, we can type the following formula into cell **B2** to do so:

=CHOOSECOLS(TEXTSPLIT(A2, " "), -1)

After entering the formula, we can then click and drag this formula down to each remaining cell in column B. Because **Microsoft Excel** uses relative cell references by default, the formula will automatically update to reference **A3**, **A4**, and so on, as it is copied down the column. This scalability is one of the primary advantages of using formula-based text manipulation. Even if your dataset grows to include thousands of rows, the extraction process remains instantaneous and accurate, drastically reducing the potential for human error associated with manual data entry.



The formula splits the names in column A based on where the space occurs and then only returns the last item from the split. As observed in the provided image, the formula successfully handles names of varying lengths. Whether the input is a simple first and last name or a more complex multi-part name, the **CHOOSECOLS** function consistently identifies the final segment. This reliability is essential for maintaining data integrity in professional environments where precision is paramount.

For this particular example, the last name of each person in column A is returned. This specific application demonstrates the practical utility of modern Excel functions in everyday business scenarios. By isolating the surname, users can perform more granular analysis, such as identifying family groupings within a list or integrating the data with other systems that require separate fields for first and last names. The efficiency gained from this automated approach allows data analysts to focus on higher-level insights rather than mundane formatting tasks.

Understanding the Logic: How This Formula Works

=CHOOSECOLS(TEXTSPLIT(A2, " "), -1)

To fully appreciate the elegance of this solution, it is helpful to break down the mechanics of how the functions interact. Here is how this formula works step-by-step. The process begins with the inner function and moves outward, a standard logical flow in **Microsoft Excel** formula evaluation. By understanding the underlying logic, you can adapt this technique to a wide variety of other data

cleaning challenges, such as extracting file extensions from paths or specific parameters from **URL** strings.

First, the **TEXTSPLIT** function splits the text in cell **A2** based on where the space occurs. The **delimiter**, defined here as a single space character enclosed in quotation marks (" "), acts as the trigger for the split. When the function encounters this character, it treats the preceding text as a single unit and the following text as the beginning of a new unit. This creates a horizontal **array** of values stored in the computer's memory, ready for the next stage of processing.

For example, the name **Andy Bernard** is split into one column that contains **Andy** and a second column that contains **Bernard**. If the input were "Andy J. Bernard", the **TEXTSPLIT** function would produce three columns: "Andy", "J.", and "Bernard". The flexibility of this function allows it to handle any number of segments, making it far superior to older methods that required the user to know the exact number of spaces in advance. It is a truly **dynamic array** behavior that adapts to the input data.

Next, the **CHOOSECOLS** function selects only the last column from the output. The syntax for **CHOOSECOLS** allows you to specify which column number(s) to return from an **array**. While you could use positive integers to select the first or second column, using negative integers allows you to reference columns from the right side of the **array**. This is a game-changer for text manipulation because it eliminates the need to calculate the total number of items in the split string.

The end result is that we're able to get only the last item from the split, which is **Bernard**. This combined formula is not only concise but also highly readable, making it easy for other users to understand your logic when they review the spreadsheet. As you continue to build more complex workbooks, prioritizing such clean and efficient formulas will contribute to the overall maintainability and performance of your Excel projects. This technique represents the best practice for modern spreadsheet design.

Note: You can find the complete documentation for the **CHOOSECOLS** function in Excel to explore further arguments and capabilities. For instance, you could use this function to return multiple specific columns or to reorder columns in a **dynamic array**, providing even greater control over your data presentation.

Exploring the TEXTSPLIT Function Parameters

While the basic usage of **TEXTSPLIT** is straightforward, the function offers several optional parameters that can enhance its utility in complex scenarios. For instance, you can specify both column and row delimiters, allowing you to transform a single string into a two-dimensional **array**. This is particularly useful when dealing with data formatted as key-value pairs or structured lists within a single cell. Mastering these parameters allows for sophisticated data restructuring without

leaving the Excel environment.

Another important parameter is the **ignore_empty** argument. In datasets where multiple spaces or redundant **delimiter** characters might exist, setting this argument to TRUE ensures that the function does not create empty columns in the resulting **array**. This ensures that when **CHOOSECOLS** looks for the last item (index -1), it finds actual data rather than a blank string. Cleaning your data at the point of extraction saves significant time in downstream processing.

The **match_mode** and **pad_with** arguments further extend the function's capabilities. Match mode allows for case-sensitive or case-insensitive delimiter matching, which can be vital when parsing technical logs or code. Padding allows you to fill missing values in a 2D array split, ensuring that your data remains aligned and structured. These professional-grade features demonstrate why **TEXTSPLIT** has become a favorite among data scientists and financial analysts who rely on **Microsoft Excel** for daily operations.

Advanced Indexing with CHOOSECOLS

The ability of **CHOOSECOLS** to accept an **array** of column indices makes it incredibly versatile. For example, if you wanted to retrieve both the first and the last item from a split string, you could use the formula **=CHOOSECOLS(TEXTSPLIT(A2, " "), 1, -1)**. This would return a two-column result containing the first and last name, effectively skipping any middle names or initials. This type of selective extraction is a common requirement in data cleansing and preparation.

Using negative indexing is a concept borrowed from advanced programming languages like Python, and its inclusion in **Microsoft Excel** signifies a shift toward more modern data handling practices. By referencing the last item as -1, the second to last as -2, and so on, you can navigate your data relative to its structure rather than its absolute position. This makes your formulas more resilient to changes in data format, such as an employee list that occasionally includes middle initials or academic titles.

Furthermore, **CHOOSECOLS** can be nested with other **dynamic array** functions like **SORT** or **FILTER**. This allows you to not only extract the last item but also perform operations on the resulting list. For instance, you could extract the last names of an entire department and then automatically sort them alphabetically, all within a single formula. This level of automation reduces the complexity of your workbooks and improves calculation speed.

Handling Multiple and Varying Delimiters

In real-world data, the **delimiter** used to separate text is not always consistent. You might encounter strings that use a mix of spaces, commas, and hyphens. Fortunately, **TEXTSPLIT** can handle multiple delimiters simultaneously by passing an **array** of characters as the second

argument. For example, using **{" ",",","-"}** would tell Excel to split the text whenever it encounters any of those three characters, providing a robust solution for messy data.

When dealing with multiple delimiters, the **ignore_empty** argument becomes even more critical. It is common for data to contain a comma followed by a space (e.g., "City, State"). If you split by both characters, Excel might interpret the space between the comma and the next word as a separate segment, resulting in empty values. By setting the function to ignore these blanks, you ensure that the "last item" retrieved by **CHOOSECOLS** is the meaningful data you expect, such as the state abbreviation.

This approach is also highly effective for parsing complex identifiers like **URL** paths. If you need to extract the final slug or file name from a long web address, you can use the forward slash (/) as a **delimiter**. Regardless of how many directories deep the file is located, **CHOOSECOLS** with a -1 index will always pinpoint the exact file name at the end of the string. This makes it an essential technique for digital marketers and web analysts.

Troubleshooting Common Errors and Limitations

While the **TEXTSPLIT** and **CHOOSECOLS** combination is powerful, users should be aware of potential errors. The most common issue is the **#VALUE!** error, which typically occurs if the **delimiter** is not found in the string or if the arguments are incorrectly formatted. It is always a good practice to wrap your formulas in an **IFERROR** function to provide a graceful fallback or a custom message when data does not meet the expected criteria.

Another consideration is the availability of these functions. As part of the **dynamic array** engine, these functions are currently available in **Microsoft Excel** for Microsoft 365 and Excel 2021 or later. If you are sharing workbooks with users on older versions of Excel, these formulas will not function, and they will see a **#NAME?** error. In such cases, you may need to rely on legacy methods or ensure that all stakeholders are using a compatible version of the software.

Finally, be mindful of the **#SPILL!** error. Although the specific formula we are discussing returns only a single value (the last item), if you were to use **TEXTSPLIT** on its own, it would attempt to populate multiple adjacent cells. If those cells already contain data, Excel cannot "spill" the results, leading to an error. Understanding the behavior of dynamic **array** outputs is key to designing clean and functional dashboards that do not break when data changes.

Best Practices for Data Cleanup and Management

To maximize the effectiveness of these functions, it is recommended to keep your source data as consistent as possible. Before applying extraction formulas, consider using the **TRIM** function to remove any leading or trailing spaces that might interfere with the **delimiter** logic. While

TEXTSPLIT is robust, starting with clean data reduces the likelihood of unexpected results and ensures that your **string** manipulations are as accurate as possible.

Additionally, documentation within your spreadsheet is vital. If you are using complex nested formulas, consider adding comments or a dedicated "Notes" column to explain the purpose of the extraction. This is especially helpful in collaborative environments where other team members may need to update or troubleshoot the workbook. Clear communication regarding how data is being parsed helps maintain the long-term integrity of the project and facilitates smoother handoffs between departments.

As you become more comfortable with **Microsoft Excel** and its dynamic functions, you will find that tasks which once took hours can now be completed in seconds. The combination of **TEXTSPLIT** and **CHOOSECOLS** is just one example of how modern tools are empowering users to handle data with greater sophistication. Continuous learning and exploration of official documentation will further enhance your proficiency and allow you to unlock the full potential of your data.

Additional Resources and Further Learning

The following tutorials explain how to perform other common operations in Excel:

How to use the TEXTBEFORE function in Excel
How to use the TEXTAFTER function in Excel
How to combine multiple cells into one with a delimiter
Advanced data sorting techniques using dynamic arrays