# PowerShell File Search Script Exercise

## Authors:

Adam Denning

Elijah Bjork

## The Big Why:

We’ve all been there. There’s that file that we definitely saved, but can’t quite remember where. A check of the common folders yields nothing, so we enter the name in File Explorer’s search bar and try our luck. We watch the green bar slowly pulse across the address bar. Surely with the length of time it is taking, File Explorer is checking every possible nook and cranny of our computer. At length, the search finishes. Nothing is returned. If only there was a simple, but effective, search program you can use...

## Objective:

This exercise guides you through the creation of a PowerShell script for searching files in a directory. You will learn how to implement administrator elevation, handle user input, and search for files based on a given term.

## Building the Script:

* Administrator Elevation: Understand how to elevate the script's privileges to administrator level.
* Reading User Input: Learn to prompt the user for the search directory and search term.
* Implementing Search Logic: Develop the logic to find files that match the search term.
* Handling Default Directory and Output: Set up the script to default to the current directory if none is specified and format the search results.

**Create .ps1:**

Create a new script named fileSearch.ps1.

**Administrator Elevation:**

To ensure we do not run into any permission errors when combing through directories, let’s make sure the script is running with administrator privileges. Copy the following lines into your script:



**Directory Prompt:**

In order to search files, we need to know where we are searching. Let’s give the user an opportunity to specify a search location, or default to the script’s location if no specified location is provided. Do the following:

* Create a Write-Host instructing the user to enter the directory to search or leave blank for current directory.
* Create a variable called $searchDirectory and populate it with a Read-Host to get the user input. This will either hold the specified directory or nothing if they chose to default.

Now we need to check if the user decided to default to the script location.

* Create an if statement to check if $searchDirectory is empty. You can do this with the condition:



* In the if statement, assign $searchDirectory to the script’s directory. (Hint: there’s an automatic variable in PowerShell that contains the directory from which the PowerShell script is being run)

**Search Prompt:**

Let’s ask the user what to search for:

* Create a Write-Host instructing the user to input a filename search term.
* Create a variable called $searchTerm and populate it with a Read-Host to get the user input.

**Search Logic:**

Now that we know where to search and what to search for, let’s create the logic for the search. There are several different ways you could approach this, so if you are feeling ambitious see if you can figure out the search on your own. Otherwise, this is one way of accomplishing the search.

* Create a function called Search-Files with the mandatory string parameters $path and $findString.
* Copy the following lines into the function. $parentPathLength will help us navigate the directories. $results will hold our results.



For this search method, we are going to treat the parent directory and subdirectories different. For the parent directory:

* Create a Write-Host that informs the user of the current path being searched. Use the $path variable.
* Copy the following lines of code that search the child items for files that match the $findString variable.



Now that those lines were spoon-fed to you, let’s search the subdirectories.

* Add a Get-ChildItem similar to the one for the parent directory, but adding -Directory after $path.
* In the ForEach-Object, create a variable called $currentPathLength and assign it the following value:



* Still in the ForEach-Object, create an if statement with an equal condition between $currentPathLength and $parentPathLength + 1.
* In the if statement, create a Write-Host that informs the user of the current path being searched, similar to the one you created earlier but this time using $($\_.FullName).
* Now, create another Get-ChildItem similar to the previous one, but instead using $\_.FullName as the path and adding -Recurse in place of -Directory. In the ForEach-Object, add the same if statement as we had in the parent directory.

The search logic should now be in place. Let’s finish off the Search-Files function by returning $results.

**Perform Search:**

Now that we have the search logic, let’s perform the search. Create a $searchResults variable and call the Search-Files function, passing along the two mandatory arguments.

**Save:**

Depending on the complexity or simplicity of the search, there could be any number of matches. Managing the results from just the terminal output might be hard, so let’s give the user the opportunity to save the results to a .txt file.

* Create a Write-Host asking the user if they would like to save the search results to a file. Indicate that they should respond with y or n.
* Create the $saveResponse variable with a Read-Host as the value to store y/n.
* Create an if statement that enters on the condition of y in the $saveResponse variable.

In the if statement:

* Create a Write-Host asking the user for the path to save the results, and indicate that they can leave blank to save to the current directory as fileSearch.txt.
* Create the $filePath variable with a Read-Host as the value.
* Add an if statement to check for an empty response and assign the default of [variable in PowerShell that contains the directory from which the PowerShell script is being run]\fileSearch.txt to the $filePath variable.
* Save the results to the file path.
* Indicate to the user that the results were saved with a Write-Host. Include the file path it got saved to.

**Exit:**

The script should now be complete. Let’s handle the script exit.

* Create a Write-Host letting the user know the search is completed. Indicate that they can press any key to exit.
* Add the following line:



## Testing the Script:

Test your script in various scenarios to ensure it works correctly. Try different directories, search terms, and see how it handles no input.

## Personalize the Script:

The beauty of creating your own search script is that it can be exactly as you want it. You can personalize the format of the results or how the results are formatted.

## Deliverable:

A PowerShell Script that locates files in a directory tree using a search term.

## Advanced Challenges:

* Add a File Type Filter: Modify the script to allow searching for specific file types.
* Error Handling: Enhance the script to gracefully handle errors and invalid inputs.