Introduction to PowerShell

a lab manual

Common setup

For all of the following lab instructions, all that you need is a recent release of Windows 10. The commands used are non-disruptive – unless very clearly marked as such – and will not make any modifications to your system.

PowerShell needs to be launched with Administrative privileges.

If you would like to keep a record of your work, you can type **Start-Transcript** at the beginning of each PowerShell session.

You may want to change the font size and coloring of your host application (Console, Terminal, ISE, or Visual Studio Code) to make text easier to read.

Exercise 1

Section 1

When working with PowerShell, you do not have to memorize all its commands. That's impossible! However, you should know how to quickly locate the right command. In this section, you will find commands by using **Get-Command -Name ****KEYWORD******** with a keyword that you should determine from the question. You do not have to run the command you find. How can you:

- Resolve a DNS name to an address.
- Suspend a print job that is printing on your printer.
- Retrieve an event log entry. Do you have to specify the log name?

Section 2

It's time to read some documentation. The best way is to use **Get-Help** ***NAME*** **-ShowWindow** where you provide the name of the command you want to find out about. If only limited help is showing, you may have to run **Update-Help** first.

Try finding out how to:

- Change the startup type of the *Print Spooler* service to "automatic".
- Display a list of network firewall rules that are enabled.
- Display the Windows edition of your current operating system.

Section 3

Modules are extensions that contain cmdlets, aliases, providers, and many other PowerShell functionalities.

- Get a list of the modules currently in memory. Run the command **Get-Volume**. Check what module was loaded into memory by comparing with the previous list.
- Locate the module that has commands for managing SMB shares.
- Find more modules (online) that deal with SMB. You don't have to install them.

Exercise 2

Section 1

Sometimes you do not need to display the entire result set of a command. Sometimes, you want to filter the result and display only certain data. Remember, that the sooner you filter, the more efficient your code.

- Display a list of the 10 newest *Security* event log entries with ID *4624*.
  Save the output to a text file.
  What would you do to show the 10 oldest events instead?
- Switch to the Certificate drive by typing **cd Cert:** (notice the colon).
  List all the objects in this location and sublocations.
  List all the certificates for which you have the private key. Results will vary.
  Export the result to a file that maintains the objects' property data types.

Exercise 3

Section 1

Scripts are typically not just run from-top-to-bottom, but rather use flow control. This includes loops and conditions.

- Create an empty string in a variable called **$Password**.
  In a **for** loop, do 8 iterations. In each iteration:
  1. Generate a random number between 65 and 122 using **Get-Random**.
  2. Convert the number to its ASCII character by casting as a **[char]** data type.
  3. Add the character at the end of the **$Password** string.

  After the loop, display the string.
- Retrieve information about your disk volumes and save it in a variable called **$Volumes**.
  Using **ForEach-Object**, for each volume, display the percentage of free space.
  Add a condition that only displays information if the volume has a *DriveLetter*.

Section 2

The internet contains a wealth of resources. Why not use them? If someone is exposing an interface into their code, you can probably consume it with PowerShell, but first you need to understand its requirements.

- Read the API documentation at https://developer.bring.com/api/postal-code/#lookup-postal-code.
  Notice the required parameters.
  Construct the request URI including the JSON URL and the required parameters.
  Make a request to the URI using **Invoke-RestMethod**.
  Display just the *Result* property of the answer.

Section 3

Be green. Reuse your code.

- Save the code from *Section 2* as a script.
  Add a parameter to the script for the postal code.
  Make the parameter mandatory. Could you give it a default value instead?
  Write documentation for your script.

Exercise 4

As you work more with PowerShell, you will build up a library of your own functions for re-use in scripts.

Section 1

It’s important to log progress, information, and errors that your script generates. Often these log files will need to time-stamp events that are written into them.

- List commands with the noun **Path**. These will be helpful.
  Create a new script file.
  At the top of the script, declare a variable called **LogFileName** with the value ‘log.txt’.
  Also, declare a function called **CreateLogPath**. After the function declaration, call the function.

Inside the function, retrieve the path of your temporary folder by accessing **$env:TEMP**.
Construct a variable called **LogFullPath** that *joins* the path of your temporary folder with **LogFileName**.
Output the **LogFileName**.
After the function call, create a blank text file at the path output by **CreateLogPath**.
Save and test your script.

- In the above script, declare a new function called **GenerateTimestamp**.
  After the function declaration, call the function.
  Inside the function, construct a variable **$LogDate** with the current date and time.
  Using the above variable, output the data in a format like *2021-12-21T12:34:56*.
- Into the log file at the location generated by **CreateLogPath**, save the current timestamp and the manufacturer and model of your computer.