Assignment: Scripting in PowerShell

Course: ID609: Operations Engineering 1,
Semester 2, 2022 Otago Polytechnic

Name: Steffen Geving Student ID: 1000103366

This assignment is worth 15%, which includes performing the actual assignment including the planning tasks as part of the instructions!

Deadline: Monday, 5 September 2022, 11:59pm

Learning Outcomes

- Configure and manage operating systems and selected services including the use of appropriate scripting languages

Instructions

In this assignment, you will face specific IT-related challenges in an organizational environment. Using a simulated business environment in which users create, modify, and delete files. Your job is to introduce a backup mechanism based on **Robocopy** that is monitored by a PowerShell script. This script sends an e-mail notification upon any addition or deletion of files (not for modified files). Events or Errors are further written to the Windows event log. Although there is a range of alternative tools that can perform those tasks out-of-the-box, we will construct them with tools that are commonly available on Windows machines.

Tools

- **Robocopy:** To establish the backup service you will be using Microsoft Robocopy (Robust File Copy), which is a standard backup tool that is highly configurable and used in small- and medium-sized companies. It is available on any current Windows environment (Go to your console prompt (Command: 'cmd') and run 'robocopy'). Since it offers a lot of options, you should be able to configure and use it as part of your IT skillset. If you use Windows at home, you may consider using it for your backup purposes.
- **PowerShell:** The core component of this work will involve building backup tools and configuration using Microsoft PowerShell. In this assignment, you will use PowerShell to integrate the functionalities provided by other tools into your solution (this is a common task in system administration using scripting). To identify newly added or deleted files and send notification e-mails and log Windows events, you will use PowerShell to parse a log file generated by Robocopy.
- **FakeSmtp:** To avoid all the nuisance that comes with varying network environments (you may do part of the assignment at home or at Polytechnic),

we use a tool that *fakes* a real SMTP mail server and does not actually send e-mails to the target address, but instead allows you to inspect sent messages. In the context of software development, e.g., when implementing e-mail sending facilities, it is a good testing tool. Changing the mail server in a production environment should then only be a matter of adjusting the configuration. When you use it, ensure you run it on a drive it can write to since it attempts to create the folder 'received-mails' where it stores mails for inspection. Once you see the user interface (as shown in Figure 1), you can start the actual server by clicking on 'Start server'. At this stage it will be able to receive e-mails. Note that relies on Java to run. More information on that is provided in the Setup section below.

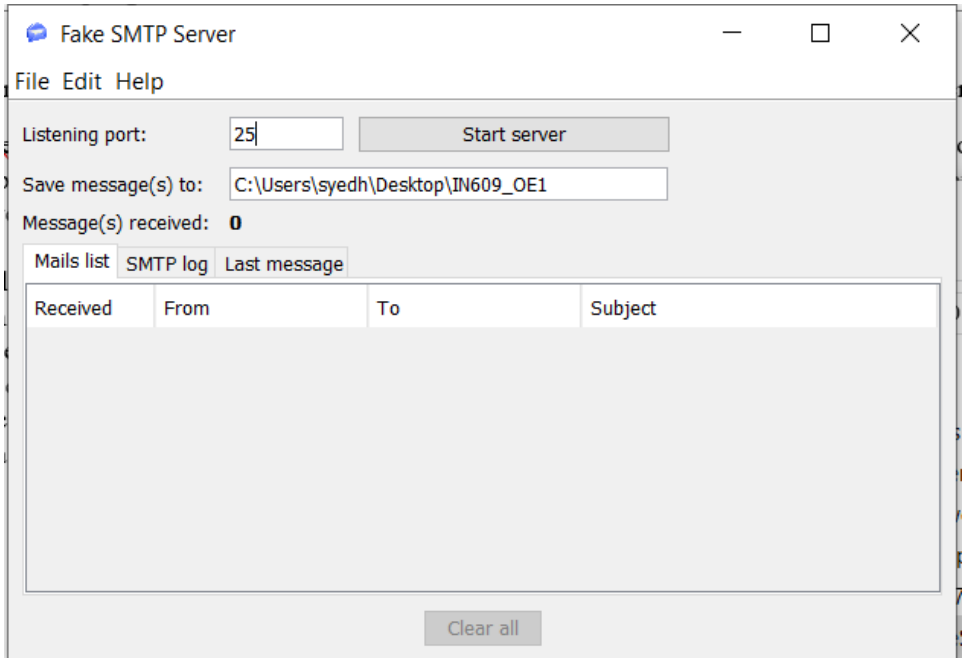

Figure 1: FakeSMTP Interface

To simulate a business environment in which users continuously create, modify and delete files you will use the tool FileModifier (see Figure 2). It offers a 'Live Mode', which will continuously perform random file operations (and which will be used to evaluate your work), and a 'Debug mode' in which you can perform specific tasks, such as performing single file modifications, creating or deleting single files.

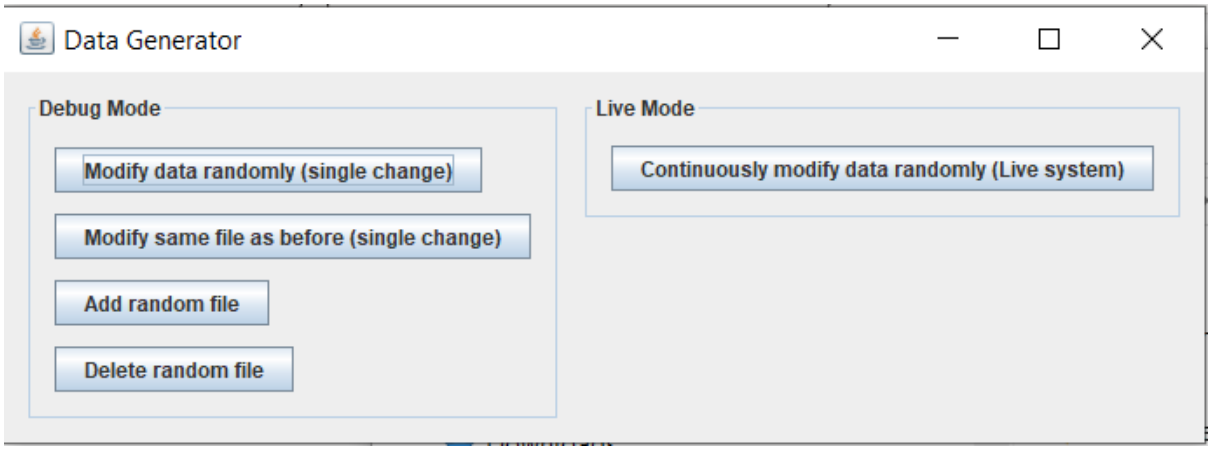

Figure 2: FileModifier Interface

Setup

To set up your environment, first, ensure you have at least PowerShell 4.0 installed. In order to run the FileModifier as well as FakeSmtp you will further require the Java Runtime Environment (https://www.java.com/en/download/manual.jsp) version 7 or higher (but you would want to use the latest version available if possible). It is already installed on OP student desktops, so you can just double-click the 'FileModifier.jar' and 'FakeSmtp-2.0.jar' and they should launch.

Overview

The entire workflow of the result of this programming task is structured as shown in Figure 3. Use this schema to plan your work packages.

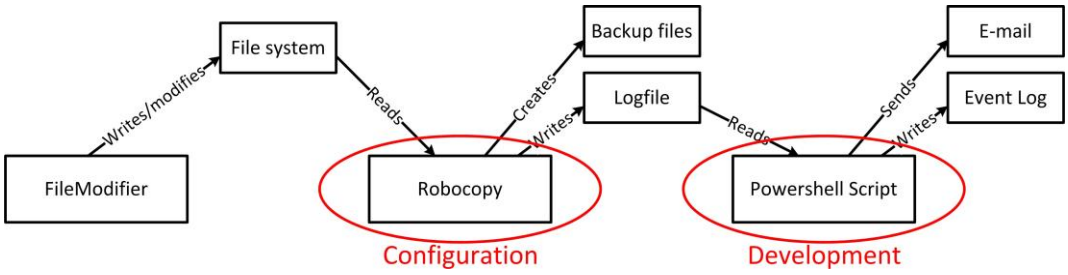

Figure 3: Task Overview

The provided FileModifier will produce a file system structure that is to be backed up using Robocopy, which produces a log file. Your PowerShell script will periodically query this logfile and check for added or deleted files. If such changes have happened, it will send an e-mail containing an overview of those changes and write those to the Windows Event log. Your work thus consists of multiple stages, with the initial task of setting up the FileModifier and configuring

Robocopy. (Note that the configuration of Robocopy is part of your submission! More on that at the end of the document.)

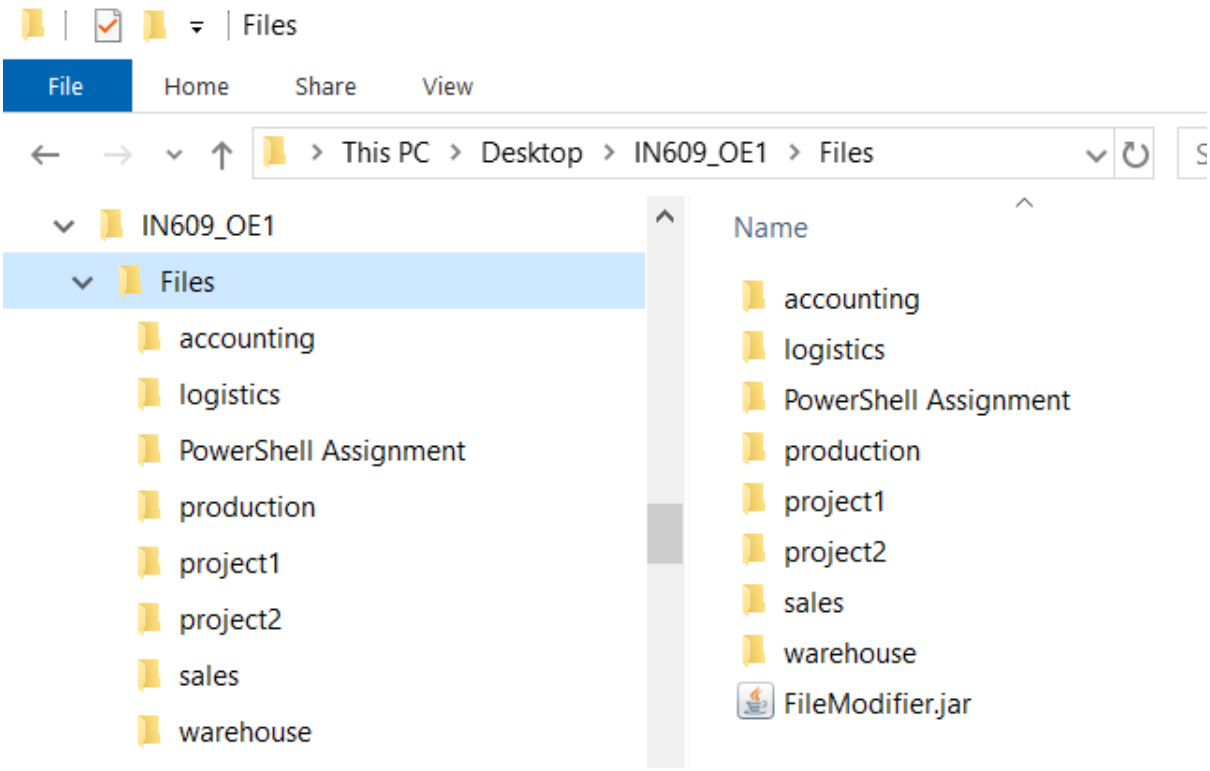

Figure 4: File System Structure

File system Structure

Set up the project file system structure, with 'C:\ID609_OE1' as the base directory (but it should be easily configurable in your code). The FileModifier will work in whatever directory it is started from. To separate the log file from the actual file system, create a subfolder called 'Files' and save the FileModifier in that folder. The structure should thus appear as shown in Figure 4. It is good practice to identify file paths as variables and store those at a single location in your code, so they can be easily changed (e.g., from C: to D: change in pathnames, etc.).

Start the FileModifier and run it in 'Live Mode' for a few minutes. You will see that it incrementally builds up the folder structure as well as files within the 'Files' subfolder. It randomly adds, deletes, and modifies files within that folder structure. You can delete the generated files at any time and start over.

Task 1: Robocopy Configuration

It is your task to set up Robocopy with the following specifications. All relevant parameters are documented in the Robocopy help (Type 'robocopy /?' on the command line). Robocopy should be configured to

- Copy the entire file structure, including all subdirectories and files of the

source directory (‘Files’) to a separate target directory (‘Files_Backup’),

- Run persistently in a separate terminal window and recheck for file system changes (Robocopy can do this out of the box – check the help),
- Write a continuous log file that contains all detected file system changes, including full pathnames of affected files. The log file should be stored in the base folder (‘ID609_OE1’) outside any of the ‘Files’ folders (‘Files’, ‘Files_Backup’).

As mentioned above, Robocopy offers an abundance of different configuration options, which you can explore by typing ‘robocopy /?’ on the command line. Other than mentioned above, no further information is required. The formatting of the actual log file is left to you, but knowing that you will need to parse it, you would want to minimize the output and remove any unnecessary information. A possible output format is in Figure 5.

```
Started : Thu Apr 23 14:45:27 2015

        *EXTRA File                 D:\OS_Temp_Target\Changes.txt
        *EXTRA File                 D:\OS_Temp_Target\Exceptions.txt
          New File                  d:\OS_Temp\accounting\data_0
          New File                  d:\OS_Temp\accounting\data_8
          New File                  d:\OS_Temp\logistics\doc_6
          New File                  d:\OS_Temp\logistics\doc_8
          New File                  d:\OS_Temp\logistics\log_1
          New File                  d:\OS_Temp\production\doc_5
          New File                  d:\OS_Temp\project1\data_3
          New File                  d:\OS_Temp\project2\doc_8
          New File                  d:\OS_Temp\project2\log_2
          New File                  d:\OS_Temp\project2\log_5
```

Figure 5: Example Log Output

For debug purposes, Robocopy further has options to write the content it saves to the log file to the screen at the same time. This way you can easily track what has been added without constantly inspecting the logfile. Carefully test your configuration until you get it right. Remember to note down the complete command, since it is necessary to replicate and test your system as part of your submission. Ideally, you save it in a batch file, so you can easily launch your complete configuration whenever you need it.

Task 2: PowerShell Scripting

You will use PowerShell to parse a log file, extract the relevant entries for new and deleted files and send a notification e-mail as well as post an event log entry in the Windows event log. This way one can follow up on events even if e-mails have been lost or deleted. The notification about specific added or deleted files should only occur once, not repeatedly (i.e., you wouldn’t want to be notified twice that file ‘C:\ID609_OE1\example.txt’ has been deleted). If something goes wrong during the execution (e.g., sending of e-mail), your script should log an error in the event log.

Initially, you would want to break the complex overall activity down into manageable work packages, which you can (to some extent) address in isolation. What are those tasks? (Write them down, so you can use them as a guide to

address the scripting tasks systematically and can revisit it later!)

Why do you think is it important to break down complex tasks into individual subtasks?

As part of the core functionality, your script is supposed to run continuously without interruption! Bear that in mind when structuring your code and consider the use of functions and loops for this purpose. Also test the setup for robustness: What happens if several components of the system do not work (e.g., robocopy, e-mail, etc.)? You may not be able to solve all required tasks at the current stage, but you know enough to get address individual subtasks immediately. We will talk about selected aspects in lectures which you can then use to tackle more complex problems before tying everything together as a final script.

General Tips

- Write your code incrementally and test it whenever you extend it. This way you can quickly identify errors. Don't write the code in one shot and test it at the end. This is a recipe for disaster
  - and my first advice would be to take it apart and test it piece by piece.
    - To achieve this, use the facilities offered by the PowerShell ISE. It offers essential features of IDEs but is much simpler than fully-fledged IDEs such as Eclipse or Netbeans. Write your scripts in the scripting pane (see Figure 7) and execute them using the 'Run Script' button (or F5 on the keyboard). The ISE further allows you to run a selection of scripts, which is useful for isolating errors when testing. To do this, mark the desired code section and click on the 'Run Selection' button (or F8).
    - Document your code. You will need to do that as part of the assignment anyway, so it would be best to do it from the beginning in order to remember what the individual statements or sections do. Especially in more complex projects, this is very important. Look at the Tutorials

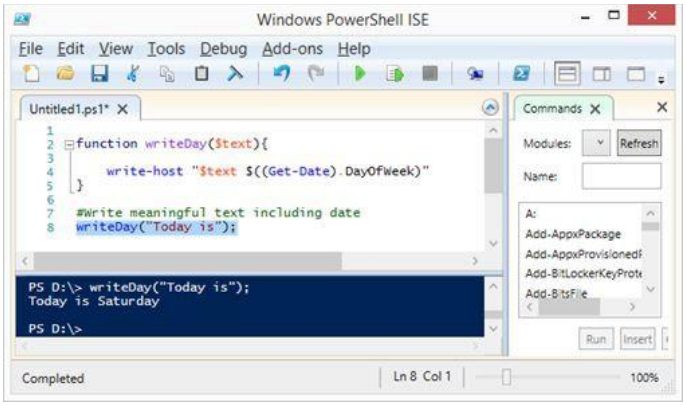

Figure 7: Example Log Output

section of the Wiki page 'PowerShell Resources', which has a great (advanced) video about how to do it right.

- It will be great to keep your code in a private git repo where you will push code as you develop.
- If you see errors, interpret them, don't just give up because you are seeing 'red'! Not all errors give you meaningful information, but they often contain a hint about what code section caused the error. This is generally a good starting point for investigation. If you really get stuck, let me know and we try to solve the issue.
  - Generally, external resources, such as file systems, e-mail, etc. can be error sources in themselves. When dealing with file systems, insufficient permission is a problem to think about, since the script may be executed with different permissions by different users. (In the first PowerShell lab we used the **'-ErrorAction'** parameter to deal with those cases.)

Testing

Test your results carefully. Your code is supposed to be generic and pretty much run on any Windows machine (with minor configuration adjustments, such as folder names). Think about possible side effects that could affect your setup, such as

- the deletion of the logfile during operation,
- the setup on different machines,
- failure of services,
- changing the location of the backup location in the script. (Changing the backup location should only require a single change in your PowerShell script!)

Deliverables

The deliverable (in addition to this worksheet) is one **zip file** with the name 'ID609S122-studentCode' (where 'studentCode' is your OP student name). This file should at least include:

- **Planning Steps** as instructed in this document (Task 2)
- **Setup Instructions** (.txt, .pdf, or .doc/x)
- **Batch file** (.bat) **or script** that starts Robocopy with the correct configuration. Ensure that you DO NOT name the file 'robocopy.bat' or 'robocopy.cmd' – else your batch file will enter an infinite loop. Any other name should work fine.
- **PowerShell script file(s)** (.ps1)
- **Screenshot of your execution**
- **Completed self-assessment sheet (as outlined below)**

Your descriptions/guide can be very brief and should be 'straight to the point', **BUT they need to be complete**. Don't make the user guess what he needs to do. The instructions should include information about the requirements of your code as well as steps or preparations necessary to get your code running. Perform the setup on a freshly restarted OP machine (or a fresh Windows VM Image) to see if the instructions are clear enough. Alternatively, you could ask someone else to set it up (not your course colleagues, though).

The PowerShell script file should contain documentation. In the header of the file, you should outline the purpose of the script, and include author name and version number. Each function should have a synopsis, description, documentation of all parameters, and an example of its usage, so it can be quickly understood.

An aspect that goes alongside documentation is formatting. **The scripts should be well formatted.** You should use indentation for conditional statements (if/else), loops and functions. Group statements if they are related (i.e. relating to the same functionality, such as parsing files). There is no definite style guide for PowerShell, but various recommendations.

Plagiarism

You can consult your fellow students to work out an issue, but you need to write your code independently. Sharing your code with others or copying code from others is not permissible and is considered plagiarism by all involved parties (i.e. even if you wrote the code and someone else copied it). If you, for some reason, used code you found on the internet (which is strongly discouraged), indicate the source and make sure you know what it does.

Deadline

The submission is due Monday, 5 September 2022, 11;59pm.

Submission Instructions

In addition to the tasks outlined above you will further complete a self-assessment on the marking sheet and indicate the tasks you completed (see column 'Completed') and may leave 'Self-assessment Comments' regarding your progress (e.g., if you did not complete a task or are unsure). (Please leave the column 'Result' and the field 'Lecturer Feedback' empty. I will fill those out.)

You will need to hand in this sheet in addition to your zip file. Please submit the zip file on the team's assignment tab before the deadline.

Feel free to ask any further submission-related questions ahead of time.

Marking Schedule (Points are 'out of 100')

	Activity		Points	Completed*	Result**
Planning	***Planning Steps***		10		
	Self-assessment Comments:				
	Lecturer Feedback:				
Task 1 – Robocopy	***Robocopy runs persistently***		5		
	Self-assessment Comments:				
	Lecturer Feedback:				
	Robocopy copies files as specified		10		
	Self-assessment Comments:				
	Lecturer Feedback:				
	Robocopy generates parseable log file		10		
	Self-assessment Comments:				
	Lecturer Feedback:				
Task 2 – PowerShell	***PowerShell script runs persistently/without errors***		10		
	Self-assessment Comments:				
	Lecturer Feedback:				
	PowerShell parses log file correctly		15		
	Self-assessment Comments:				
	Lecturer Feedback:				

* Tick if completed, O if some completed, and cross if not done. You can leave comments in the field below (e.g. aspects you did not finish, etc.).

** Do not fill anything here. This field is used by the lecturer.

	Activity	Points	Completed*	Result**
Task 2 – PowerShell	***Filtering new and deleted files***	10		
	Self-assessment Comments:			
	Lecturer Feedback:			
	Sending e-mail notifications	7.5		
	Self-assessment Comments:			
	Lecturer Feedback:			
	Adding EventLog entry	7.5		
	Self-assessment Comments:			
	Lecturer Feedback:			
Documentation & Code Quality	***Commenting***	5		
	Self-assessment Comments:			
	Lecturer Feedback:			
	Code Quality	5		
	Self-assessment Comments:			
	Lecturer Feedback:			
	Setup Instructions	5		
	Self-assessment Comments:			
	Lecturer Feedback:			

* Tick if completed, O if some completed, and cross if not done. You can leave comments in the field below (e.g. aspects you did not finish, etc.).

** Do not fill anything here. This field is used by the lecturer.