

CertHE Computing Skills for Workplace
Module: Computer Architecture and Operating Systems
Module Code: ACCA4022

Operating Systems Assignment

27 April 2026

Table of Contents

Introduction	1
Task 1: Key functions of general Operating Systems.....	2
Task 2 – Use of Windows OS CLI Commands	3
Task 3 – Use of Linux OS CLI Commands.....	5
Task 4 – Use of Windows OS Scripts.....	7
Task 5 – Use of Linux OS Scripts.....	10
Conclusion.....	12
Reference List	12
Appendices.....	13
Appendix 1 – Tutorial Activity: On OS Types (from OS Tutorial 1)	13
Appendix 2 – Lecture Activities / Activity Log.....	15
Appendix 3 – Evidence of Engagement with Online Certified Course	16

Introduction

Operating systems form the foundation of modern computing, acting as the essential layer between hardware and the applications that users rely on every day for work, study and play. This report examines the core functions that allow general-purpose operating systems to manage resources efficiently and provide a stable platform for software execution. My report then moves on to show practical demonstrations of command-line tools on both Windows and MacOS/Unix-based systems, showing how these tools are used for everyday system administration tasks such as monitoring processes, managing files, testing networks, and scheduling jobs. Towards the end, my report will present four simple automation scripts that illustrate how basic programming techniques can be used in order to create useful system utilities. Through these sections, the report aims to demonstrate both theoretical understanding and practical competence in working with operating systems.

Task 1: Key functions of general Operating Systems

General-purpose operating systems such as Windows 11, Ubuntu Linux, and macOS serve as the critical bridge between computer hardware and the software that users interact with daily. [3] Their main purpose is to manage the system's limited resources in a fair and efficient way while hiding the complexity of the underlying hardware from both users and applications.

Arguably the most important function that an Operating System carries out is process management. The operating system is responsible for creating, scheduling, and terminating processes in its environment. In a Windows environment, this can be observed clearly in Task Manager, where users can see which programs are consuming the most CPU, system memory and storage usage. On Linux and macOS, tools such as `ps` and `top` provide similar real-time information. The operating system uses scheduling algorithms [9] to decide which process gets CPU time next [1], optimizing tasks to ensure that no single program can monopolise the processor which could cause the user to perceive poor performance from the system.

Memory management is equally vital for an Operating System to be an effective solution. A computer has finite memory at all times. The operating system must therefore be effective at allocating and deallocating the system's available RAM memory to currently running programs and processes to ensure overall system responsiveness and efficiency of the computer. Some techniques that Windows uses are virtual memory techniques, such as paging to backing storage to prevent one application from crashing the entire system – this is called a page-file [1] and it can be turned off entirely although not recommended on machines with low RAM capacity. Windows users can monitor this through Resource Monitor [9], while Linux users can use the simple `free -h` command to see memory usage at a glance.

Another key role an operating system plays is file and device management. The operating system organises data into a structured file system that it can easily parse and index quickly for split-second retrieval (NTFS on Windows, ext4 on Linux) and controls access to attached storage devices, printers, and other peripherals like cameras, keyboards, speakers and display. It also enforces security and access control [2] through user accounts, permissions, and features such as User Account Control (UAC) on Windows or file permissions on Linux. It is the main area of focus in cyber-security professions and is the same mechanism that allows and denies certain users and groups access to files and/or software.

Finally, operating systems provide both graphical and command-line interfaces and handle networking tasks, allowing computers to communicate with each other and the internet. Graphical interfaces refer largely to point-and-click software solutions which became mainstream in the late 90's, consequently widening accessibility of everyday computing to people who had little understanding of computer code. Command line interfaces tend to be used more by power-users nowadays like network administrators, who may need to scale the same code over networked machines without being physically present at the machine – Powershell has proved itself very useful for me in the setting of a multi-academy-trust of schools for exactly this reason.

Together, these collective functions create the smooth, multi-tasking experience that users expect from modern computers. These functions work together to create the illusion of a responsive, multi-tasking environment even on shared hardware resources. [8] Without an Operating System, users would need to program directly against raw hardware, which is not practical or reasonable for everyday use.

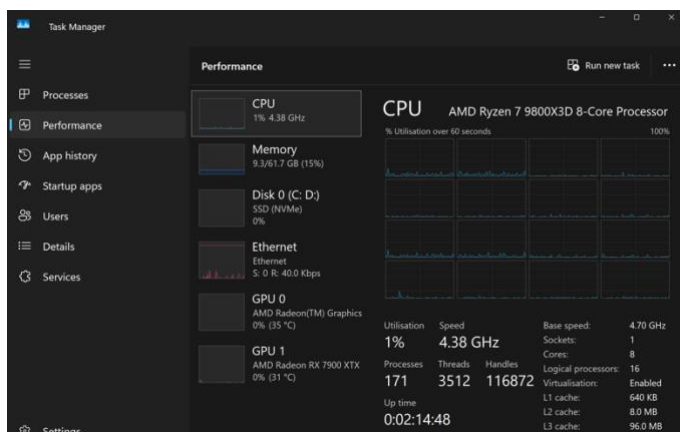


Figure 1: Screenshot of Windows Task Manager showing process and performance monitoring.

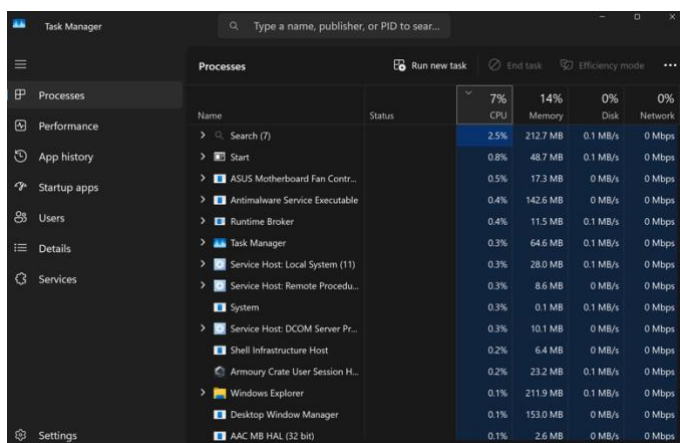


Figure 2: Task Manager listing all processes

Task 2 – Use of Windows OS CLI Commands

2. Use of Windows OS CLI Commands

This section demonstrates twelve practical commands available in Windows 11, using both Command Prompt and PowerShell. The commands have been organised into five categories that reflect common system administration tasks: process management, file and directory management, network testing, user management, and task scheduling. All examples were run on a Windows 11 system.

Process Management The `Get-Process` cmdlet in PowerShell is particularly useful for identifying which applications are consuming the most system resources.

Running `Get-Process | Sort-Object CPU -Descending | Select-Object -First 10` displays the top ten processes sorted by CPU usage.

If a process needs to be terminated, the `Stop-Process` cmdlet can be used, for example `Stop-Process -Name notepad -Force`. [4]

File and Directory Management

PowerShell is also capable of robust file management techniques. When I used the command `Get-Childitem -Path C:\Users\Ryan\Documents -Recurse | Where-Object {$_.Length -gt 1MB}` it displayed any files larger than one megabyte within the Documents folder.

`New-Item -Path "C:\Backup" -ItemType Directory` quickly creates new directories.

Network Testing and Monitoring

I was able to test network connectivity using the `Test-Connection` cmdlet. For example, `Test-Connection google.com -Count 4` sends four ping requests to Google's server and displays the results.

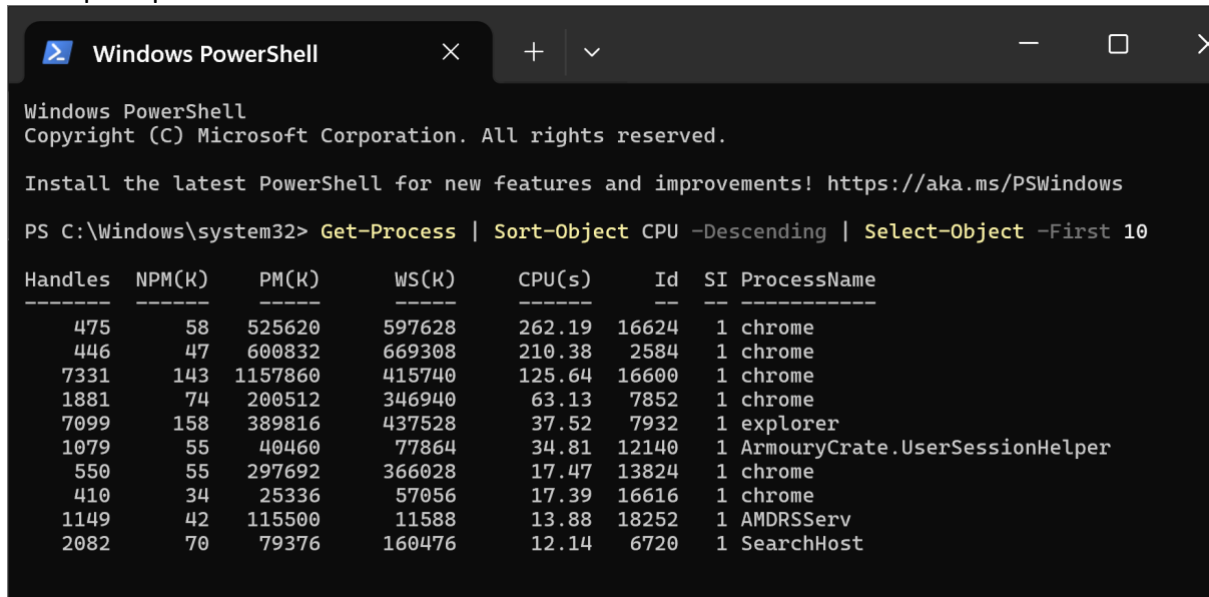
To view all IPv4 addresses that have been configured on the system, the command `Get-NetIPAddress | Where-Object {$_.AddressFamily -eq "IPv4"}` can be used.

User Management

Local user accounts can be managed directly from PowerShell. When I used the command `Get-LocalUser | Where-Object {$_.Enabled -eq $true}` it produced a list of all enabled local accounts, whilst `New-LocalUser` can be used to create new accounts (this however required administrative privileges).

Task Scheduling

Windows includes a built-in task scheduler that can be controlled from the command line. The command `Get-ScheduledTask | Where-Object {$_.State -eq "Ready"} | Select-Object TaskName, State -First 5` gave me a list of the first five scheduled tasks that are ready to run. I was able to schedule new tasks by using the `schtasks` utility, such as scheduling a daily backup script.



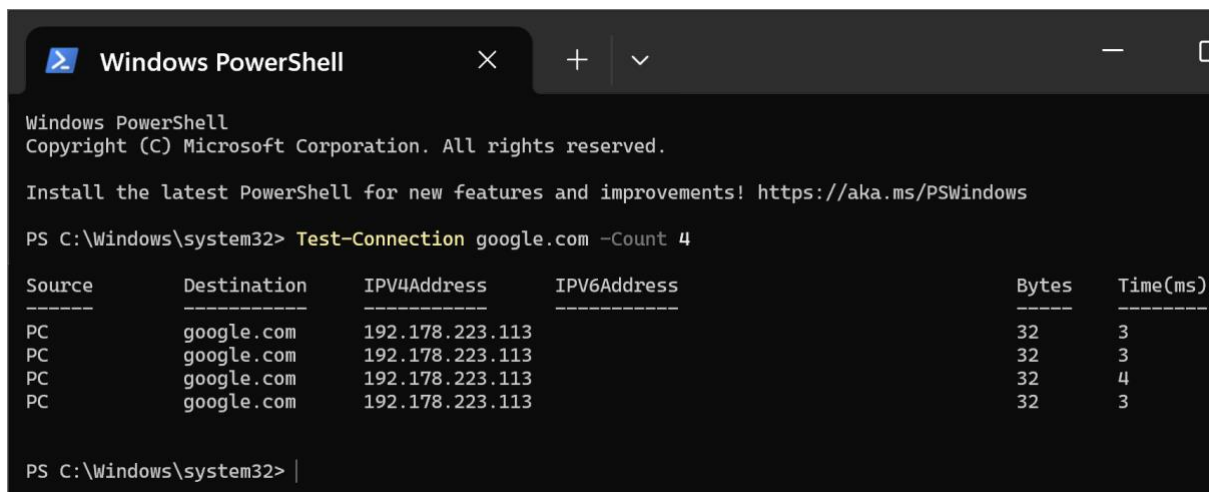
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> Get-Process | Sort-Object CPU -Descending | Select-Object -First 10
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
475	58	525620	597628	262.19	16624	1	chrome
446	47	600832	669308	210.38	2584	1	chrome
7331	143	1157860	415740	125.64	16600	1	chrome
1881	74	200512	346940	63.13	7852	1	chrome
7099	158	389816	437528	37.52	7932	1	explorer
1079	55	40460	77864	34.81	12140	1	ArmouryCrate.UserSessionHelper
550	55	297692	366028	17.47	13824	1	chrome
410	34	25336	57056	17.39	16616	1	chrome
1149	42	115500	11588	13.88	18252	1	AMDRSServ
2082	70	79376	160476	12.14	6720	1	SearchHost

Figure 3: Screenshot of PowerShell showing `Get-Process` output (top processes by CPU).



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Windows\system32> Test-Connection google.com -Count 4
```

Source	Destination	IPV4Address	IPV6Address	Bytes	Time(ms)
PC	google.com	192.178.223.113		32	3
PC	google.com	192.178.223.113		32	3
PC	google.com	192.178.223.113		32	4
PC	google.com	192.178.223.113		32	3

```
PS C:\Windows\system32> |
```

Figure 4: Screenshot of Command Prompt showing `Test-Connection` results.

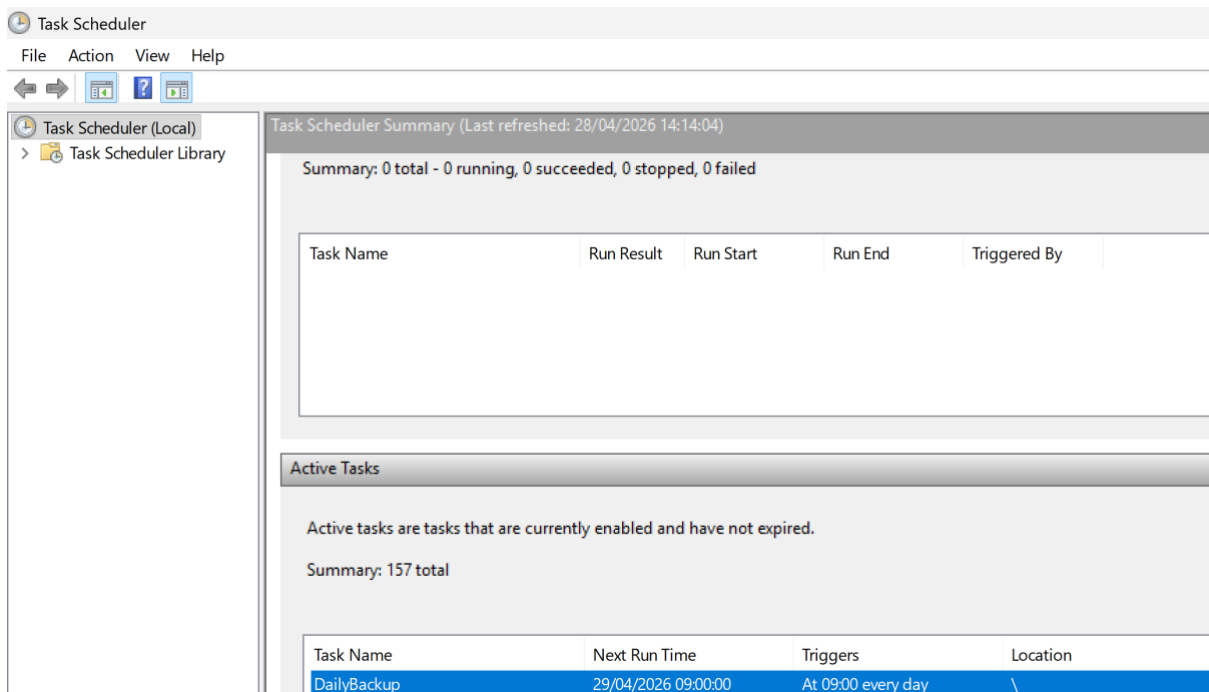


Figure 5: Screenshot of Task Scheduler GUI with newly created “DailyBackup” task.

These commands illustrate how Windows provides a rich set of command-line tools [11] that allow users to monitor and control the system efficiently without having to rely solely on graphical interfaces. This is particularly useful when trying to manage multiple endpoints in one sweep. [6]

Task 3 – Use of Linux OS CLI Commands

3. Use of Linux OS CLI Commands

This section presents twelve practical Bash commands that can be used on Linux distributions (distros) and macOS systems. In my case, I was using MacOS Monterey 12.7.6. The commands have been grouped into the same five categories used in the Windows section to allow the reader easy comparison between the two platforms. All examples were executed in Terminal on macOS, which shares the same Unix foundations as Linux.

Process Management

When I used the `ps aux | head -10` command, I was returned quick snapshot of the top ten running processes, including their current CPU and memory usage.

For a more interactive view, the `top` command can be used (pressing `q` to exit).

If I needed to stop a process, `kill -9` followed by the process ID, which is found in Activity Monitor, will terminate that specified application immediately.

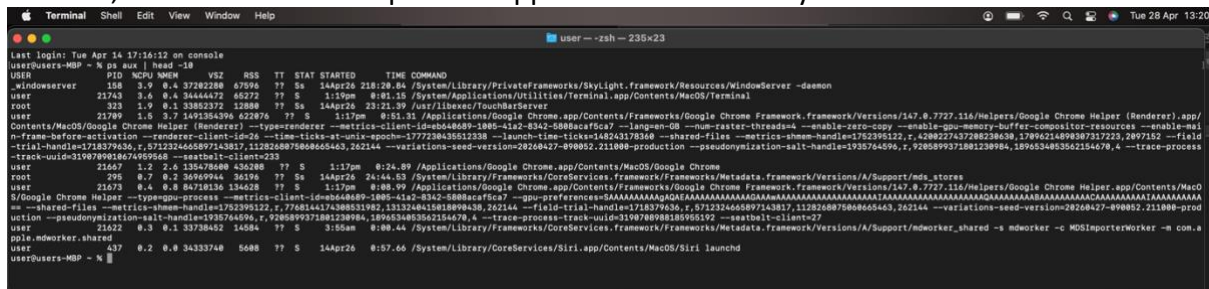


Figure 6: Screenshot of Terminal showing `ps aux | head -10` output.

Figure 7: Screenshot of Terminal command `htop` showing a more interactive view of currently running processes

File and Directory Management

Basic file operations are handled efficiently in the terminal. [10] When I enter command `ls -la ~/` Terminal lists all files and directories in the home folder with detailed permission information for each file in that location.

I can create new directories created using `mkdir -p ~/OS_Assignment`, and I can check disk space with `df -h` or `du -sh ~/Documents`.

Figure 8: Screenshot of `df -h` command displaying disk usage.

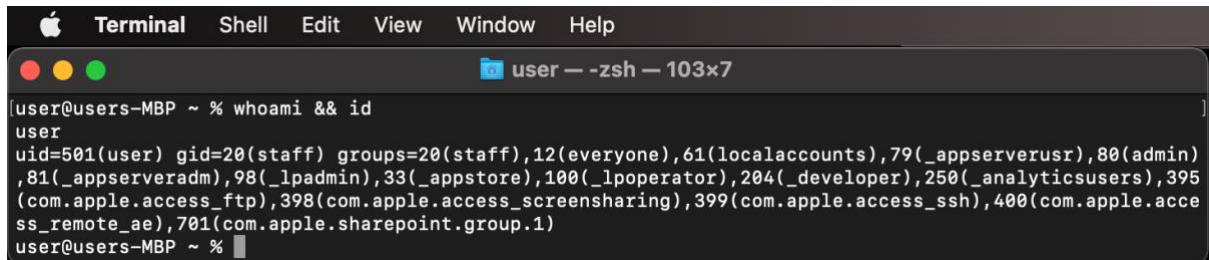
Network Testing and Monitoring

I then tested network connectivity using the basic `ping -c 4 google.com`, which sends four data packets to Google's server and reports the round-trip times to ensure that the connection is stable. To view the IPv4 addresses assigned to network interfaces, I can use the command `ifconfig | grep "inet "`.

Figure 9: Screenshot of `ping -c 4 google.com` results.

User Management

The currently logged in user's identity is outputted when using the command `whoami && id`. The output shows both the username, display name and group IDs associated with the account.



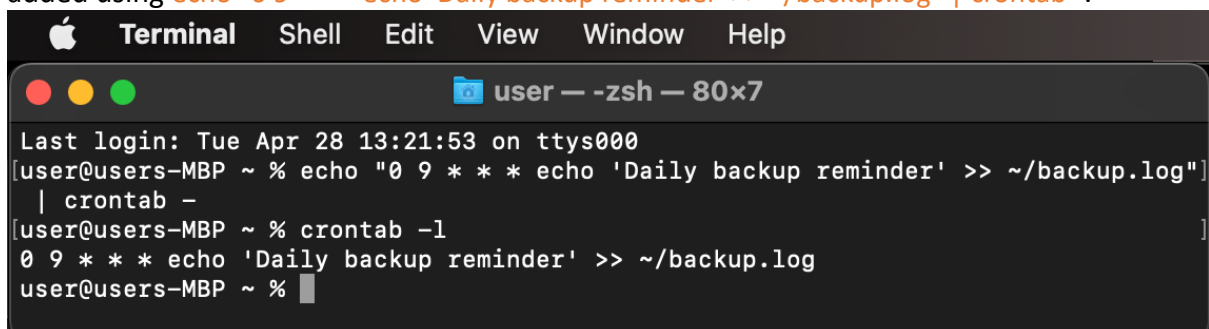
```
Terminal Shell Edit View Window Help
user — -zsh — 103x7
[user@users-MBP ~ % whoami && id
user
uid=501(user) gid=20(staff) groups=20(staff),12(everyone),61(localaccounts),79(_appserverusr),80(admin),81(_appserveradm),98(_lpadmin),33(_appstore),100(_lpoperator),204(_developer),250(_analyticsusers),395(com.apple.access_ftp),398(com.apple.access_screensharing),399(com.apple.access_ssh),400(com.apple.access_remote_ae),701(com.apple.sharepoint.group.1)
user@users-MBP ~ %
```

Figure 10: Screenshot of `whoami && id` command output.

Task Scheduling

Recurring tasks are managed through cron. [7]

The command `crontab -l` lists all scheduled jobs for the current user. A new daily job can be added using `echo "0 9 * * * echo 'Daily backup reminder' >> ~/backup.log" | crontab -`.



```
Terminal Shell Edit View Window Help
user — -zsh — 80x7
Last login: Tue Apr 28 13:21:53 on ttys000
[user@users-MBP ~ % echo "0 9 * * * echo 'Daily backup reminder' >> ~/backup.log" | crontab -
[user@users-MBP ~ % crontab -l
0 9 * * * echo 'Daily backup reminder' >> ~/backup.log
user@users-MBP ~ %
```

Figure 11: Screenshot of `crontab -l` after adding the daily backup job.

These commands illustrate how Linux (and macOS) provide lightweight, powerful Command Line Interface (CLI) tools for system administrators, which from experience can be particularly helpful when aiming to scale across a network. The combination of process control, file system inspection, network diagnostics, and cron scheduling enables robust, efficient automation and monitoring [5] [10] without the requirement of a graphical interface.

Task 4 – Use of Windows OS Scripts

4. Use of Windows OS scripts

To demonstrate automation using Windows command-line tools, I created two scripts. Both show that I can competently create usable scripts which incorporate variables, user input, conditional logic, and basic arithmetic or string operations, reflecting real-world administrative tasks.

Script 1: System Information & Disk Checker (Batch)

The first script I created, written in Batch and not to be confused with VBScript, gathers basic system information and optionally checks available disk space on the C: drive. [4] It uses variables to store the user's name and the free space value, then employs an if statement to decide whether to perform the disk check based on user input. I think this type of script could prove useful for quick system health checks or as part of a larger maintenance routine, before running `DISM` and `sfc-scannow`.

```

@echo off
title Simple Backup Demo Script

echo =====
echo   Simple Backup Demo Script
echo =====

set /p username=Enter your name:
echo.
echo Hello, %username%!

echo.
set /p source=Enter folder to backup (e.g. Desktop):

set /p dest=Enter backup folder name:

set timestamp=%date:~-4%%date:~3,2%%date:~0,2%_%time:~0,2%%time:~3,2%%time:~6,2%
set timestamp=%timestamp: =0%

mkdir "%dest%" 2>nul

echo.
echo Creating backup of %source%...

xcopy "%source%\*.*" "%dest%\%timestamp%" /E /I /Q >nul 2>&1

if %errorlevel% equ 0 (
    echo.
    echo Backup successful!
    echo Folder created: %dest%\%timestamp%
) else (
    echo.
    echo Backup failed. Check folder name.
)

echo.
echo =====
echo Script completed successfully.
echo =====
echo.
pause

```

```

backscript.bat
File Edit View
set /p dest=Enter backup folder name:
set timestamp=%date:~-4%%date:~3,2%%date:~0,2%_%time:~0,2%%time:~3,2%%time:~6,2%
set timestamp=%timestamp: =0%
mkdir "%dest%" 2>nul
echo.
echo Creating backup of %source%...
xcopy "%source%\*.*" "%dest%\%timestamp%" /E /I /Q >nul 2>&1
if %errorlevel% equ 0 (
    echo.
    echo Backup successful!
    echo Folder created: %dest%\%timestamp%
) else (
    echo.
    echo Backup failed. Check folder name.
)
echo.
echo =====
echo Script completed successfully.
echo =====
echo.
pause

```

```

Simple Backup Demo Script
=====
Enter your name: Ryan
Hello, Ryan!
Enter folder to backup (e.g. Desktop): Desktop
Enter backup folder name: Documents
Creating backup of Desktop...
Backup failed. Check folder name.
=====
Script completed successfully.
=====
Press any key to continue . . . |

```

Script 2: Process Monitor & Killer (PowerShell)

The second script, which I wrote for use in PowerShell, displays the top five processes by CPU usage and allows the user to terminate a process by entering its ID. It uses a 'try-catch' block to handle errors gracefully, such as when an invalid process ID is entered by the user. This demonstrates how PowerShell's object-oriented approach [11] can be used to create more robust and user-friendly administrative tools.

PowerShell

```
Write-Host "=== PowerShell Process Manager ===" -ForegroundColor Cyan
```

```
$name = Read-Host "Enter your name"
Write-Host "Hello, $name!"
```

```
$processes = Get-Process | Sort-Object CPU -Descending | Select-Object -First 5
Write-Host "`nTop 5 CPU processes:"
$processes | Format-Table Name, CPU, Id -AutoSize
```

```
$kill = Read-Host "`nDo you want to kill a process? (Y/N)"
if ($kill -eq "Y") {
    $pidToKill = Read-Host "Enter Process ID to terminate"
    try {
        Stop-Process -Id $pidToKill -Force -ErrorAction Stop
        Write-Host "Process $pidToKill terminated successfully." -ForegroundColor Green
    } catch {
        Write-Host "Error: Invalid Process ID or insufficient permissions." -ForegroundColor Red
    }
} else {
    Write-Host "No process terminated."
}
```

```
Write-Host "`nScript completed. Goodbye, $name!" -
ForegroundColor Green
```

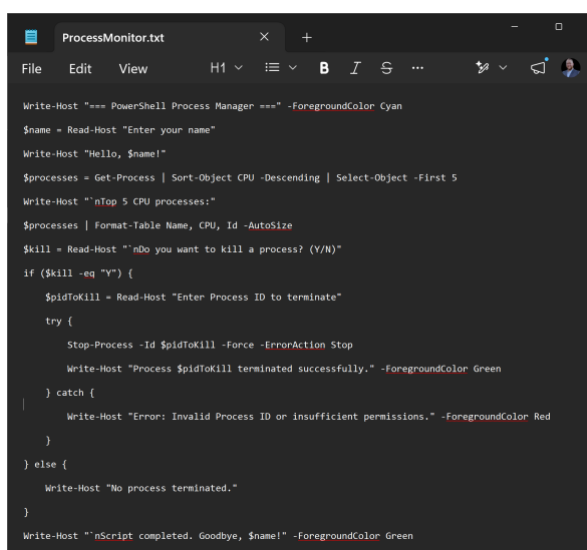


Figure 14: Screenshot of the PowerShell script code.

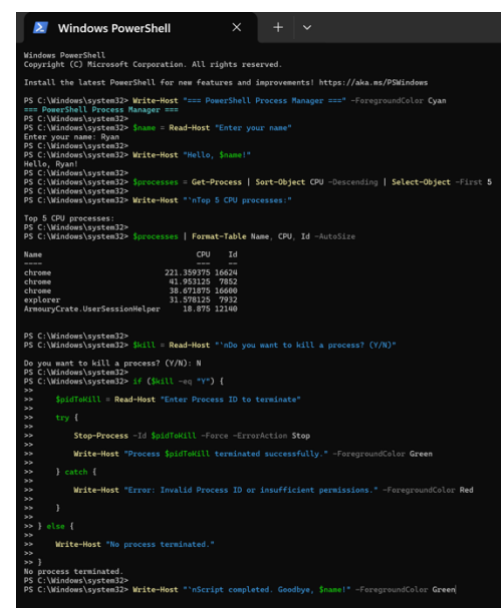


Figure 15: Screenshot of the script executing with process list and user interaction.

5. Use of Linux OS scripts

I then created two Bash scripts on my Macbook Pro, with the intention to illustrate automation on Linux and macOS systems. Both scripts show I can competently make use of variables, user input via read, conditional statements, and command substitution to produce useful output.

The first script I created collects crucial system information like hostname, kernel version, uptime, and memory usage. It then asks the user whether they would like to see disk usage and displays the result only if they respond with a “Y” keystroke.

This shows how simple scripts can be made interactive and user-friendly.

Script 1: System Info & Memory Reporter (Bash)

```
#!/bin/bash

echo "=====
echo "   Linux/macOS System Info Script"
echo "=====

read -p "Enter your name: " name
echo "Hello, $name!"

echo ""
echo "System Information:"
echo "Hostname: $(hostname)"
echo "Kernel: $(uname -r)"
echo "Uptime: $(uptime -p)"

echo ""
total_mem=$(free -h | awk '/Mem:/ {print $2}')
used_mem=$(free -h | awk '/Mem:/ {print $3}')
echo "Memory: Total = $total_mem | Used = $used_mem"

read -p "Would you like to see disk usage? (y/n): " choice
if [[ "$choice" == "y" || "$choice" == "Y" ]]; then
    echo ""
    df -h | head -5
else
    echo "Disk check skipped."

fi
echo ""
echo "Script completed successfully. Thank you, $name!"
```

```

# System Info Script in Bash

#!/bin/bash

echo "=====
Linux/macOS System Info Script
=====

read -p "Enter your name: " name
echo "Hello, $name!"

echo ""
echo "System Information:"
echo "Hostname: $(hostname)"
echo "Kernel: $(uname -r)"
echo "Uptime: $(uptime -p)"

echo ""
total_mem=$(free -h | awk 'Mem/{print $2}')
used_mem=$(free -h | awk 'Mem/{print $3}')
echo "Memory: Total = $total_mem | Used = $used_mem"

read -p "Would you like to see disk usage? (y/n): " choice
if [[ "$choice" == "y" || "$choice" == "Y" ]]; then
    echo ""
    df -h | head -5
else
    echo "Disk check skipped."
fi

echo ""
echo "Script completed successfully. Thank you, $name!"

```

Figure 16: Screenshot of the first Bash script in a text editor.

```

user@users-MBP ~ % nano ~/system_info.sh
user@users-MBP ~ % chmod +x ~/system_info.sh
user@users-MBP ~ % ~/system_info.sh

=====
Linux/macOS System Info Script
=====

Enter your name: Ryan
Hello, Ryan!

System Information:
Hostname: users-MBP
Kernel: 21.6.0
uptime: illegal option -- p
usage: uptime
Uptime:

/Users/user/system_info.sh: line 17: free: command not found
/Users/user/system_info.sh: line 18: free: command not found
Memory: Total = | Used =
Would you like to see disk usage? (y/n): Y

Filesystem      Size  Used Avail Capacity  iused   ifree %used   Mounted on
/dev/disk1s1s1  932Gi  14Gi  804Gi    2%  602388 4293964784    0%   /
devfs           197Ki  197Ki   8Bi  100%    682      0  100%   /dev
/dev/disk1s4    932Gi  3.6Gi  804Gi    1%    4  8427867120    0%   /System/Volumes/VW
/dev/disk1s2    932Gi  254Mi  804Gi    1%   815  8427867120    0%   /System/Volumes/Preboot

Script completed successfully. Thank you, Ryan!

=====
Linux/macOS System Info Script
=====

Enter your name: ^[

```

Figure 17: Screenshot of the first script running in Terminal with user input and bash output.

Script 2: Automated Backup with Date & Size (Bash)

The second script I created performs a fairly basic backup operation. It prompts the user for a source folder and destination for the backup copy to go to, creates a timestamped compressed archive using tar, and reports the size of the resulting backup file. I have included an error handling feature which informs the user if the backup fails. On reflection, this script could very easily be expanded or scheduled using cron for regular automated backups. A more advanced change to the script that I considered could be on it's error handling aspect, where it could give the user a reason why the backup failed so they can mitigate and try again. This would improve usability a lot.

```

#!/bin/bash

echo "=== Simple Backup Demo Script ==="

read -p "Enter your name: " name
echo "Hello $name!"

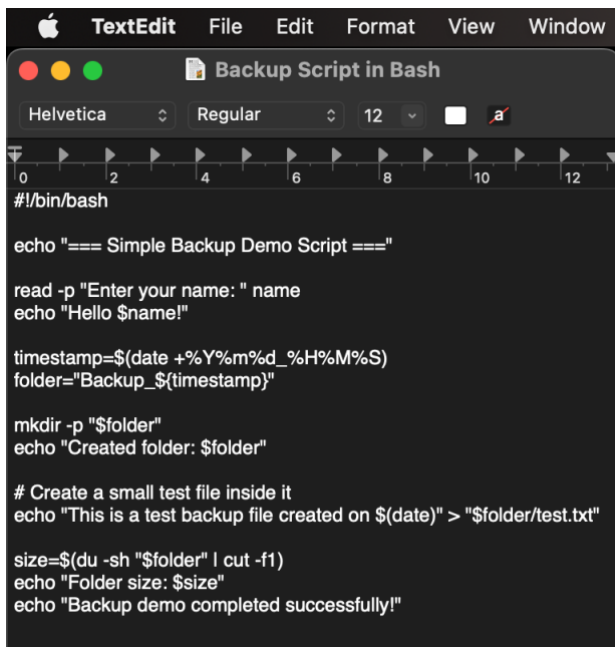
timestamp=$(date +%Y%m%d_%H%M%S)
folder="Backup_${timestamp}"

mkdir -p "$folder"
echo "Created folder: $folder"

# Create a small test file inside it
echo "This is a test backup file created on $(date)" > "$folder/test.txt"

size=$(du -sh "$folder" | cut -f1)
echo "Folder size: $size"
echo "Backup demo completed successfully!"

```



```
#!/bin/bash

echo "=== Simple Backup Demo Script ==="

read -p "Enter your name: " name
echo "Hello $name!"

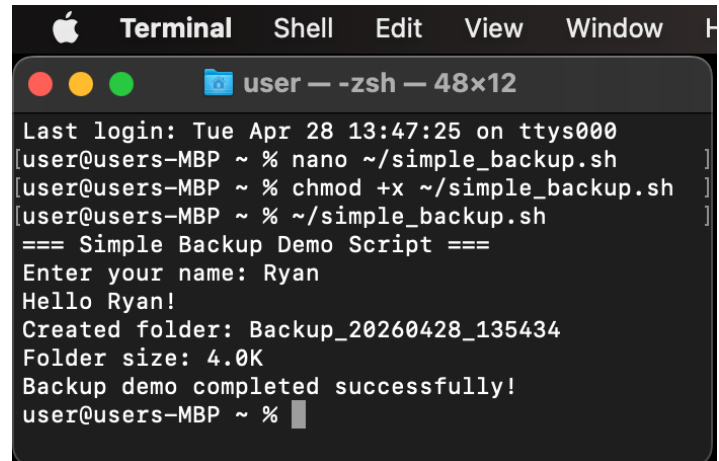
timestamp=$(date +%Y%m%d_%H%M%S)
folder="Backup_${timestamp}"

mkdir -p "$folder"
echo "Created folder: $folder"

# Create a small test file inside it
echo "This is a test backup file created on $(date)" > "$folder/test.txt"

size=$(du -sh "$folder" | cut -f1)
echo "Folder size: $size"
echo "Backup demo completed successfully!"
```

Figure 18: Screenshot of the second Bash script code.



```
user — -zsh — 48x12

Last login: Tue Apr 28 13:47:25 on ttys000
[user@users-MBP ~ % nano ~/simple_backup.sh
[user@users-MBP ~ % chmod +x ~/simple_backup.sh
[user@users-MBP ~ % ~/simple_backup.sh
=== Simple Backup Demo Script ===
Enter your name: Ryan
Hello Ryan!
Created folder: Backup_20260428_135434
Folder size: 4.0K
Backup demo completed successfully!
user@users-MBP ~ %
```

Figure 19: Screenshot of the script running and creating a compressed backup with size output.

Conclusion

I believe this report has explored both the theoretical foundations and practical applications of modern operating systems thoroughly, and it has helped me appreciate the inner-workings of an OS that we no longer see due to General User Interfaces (GUI's). The key functions I outlined in the first section which discusses process and memory management, file systems, security, and user interfaces all form the essential framework that allows modern computers to operate in the ways that we have come to expect them to, reliably and efficiently.

The command-line demonstrations on Windows and Linux/macOS showed how these functions can be monitored and controlled directly by the user or administrator, providing a granular level of precision and speed that graphical interfaces often cannot match, and can usually be scaled at network level, which helps with scalability.

Lastly, the four scripts that I showcase in the later sections illustrates how even relatively simple programming can create practical automation tools that save time and reduce the risk of human error, which is important for mission-critical environments.

Overall, the assignment has reinforced the value of developing strong command-line skills alongside theoretical knowledge, skills that I am sure will prove essential in both future academic work and my professional IT role as IT infrastructure technician.

Reference List

- [1] A. Silberschatz, P. B. Galvin, and G. Gagne, **Operating System Concepts**, 10th ed. Hoboken, NJ, USA: Wiley, 2018.
- [2] W. Stallings, **Operating Systems: Internals and Design Principles**, 9th ed. Harlow, UK: Pearson, 2018.

- [3] GeeksforGeeks, "Functions of Operating System," 2025. [Online]. Available: <https://www.geeksforgeeks.org/operating-systems/functions-of-operating-system/>
- [4] Microsoft, "PowerShell," *Microsoft Learn*, 2026. [Online]. Available: <https://learn.microsoft.com/en-us/powershell/>
- [5] The Linux man-pages project, "Linux man pages," *man7.org*, 2025. [Online]. Available: <https://man7.org/linux/man-pages/>
- [6] Microsoft, "PowerShell scripting," *Microsoft Learn*, 2025. [Online]. Available: <https://learn.microsoft.com/en-us/powershell/scripting/>
- [7] GeeksforGeeks, "Job Scheduling Commands in Linux," Dec. 2025. [Online]. Available: <https://www.geeksforgeeks.org/linux-unix/job-scheduling-commands-in-linux/>
- [8] Coursera, "5 Types of Operating Systems to Know," Jan. 2026. [Online]. Available: <https://www.coursera.org/articles/types-of-operating-systems>
- [9] Red Hat, "Linux System Administration," Red Hat Documentation, 2025. [Online]. Available: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/
- [10] Tanenbaum, A. S. and Bos, H., Modern Operating Systems, 4th ed. Harlow, UK: Pearson, 2015.
- [11] Netwrix, "Essential PowerShell Commands: A Cheat Sheet," Jan. 2025. [Online]. Available: <https://netwrix.com/en/resources/blog/powershell-commands-cheat-sheet/>

Appendices

Appendix 1 – Tutorial Activity: On OS Types (from OS Tutorial 1)

As part of OS Tutorial 1, I conducted a detailed comparative analysis of five major operating systems. The purpose of this activity was to develop a deeper understanding of how different operating systems are designed to meet varying user needs, hardware constraints, and security requirements. The comparison focused on technical architecture, file system design, security models, user experience, and real-world applications.

Comparative Analysis of Operating Systems

Operating System	Kernel Architecture	File System	Memory Management Model	Multitasking Model	Security & Access Control	User Interface	Primary Target Audience	Key Strengths	Key Limitations
Windows 11	Hybrid (NT Kernel)	NTFS	Virtual Memory + Paging	Pre-emptive Multitasking	UAC, BitLocker, Windows Defender, AppContainer	Graphical (Desktop)	General users, Gamers, Enterprises	Excellent backward compatibility, Strong gaming ecosystem, Rich software library	Higher susceptibility to malware, Resource-heavy, Expensive licensing
Linux (Ubuntu)	Monolithic (Linux Kernel)	ext4 (default)	Virtual Memory + Paging + Swapping	Pre-emptive + Cooperative	POSIX permissions, SELinux/AppArmor, Firewall	CLI + GUI (GNOME/KDE)	Developers, System Administrators, Servers	Highly customisable, Extremely stable, Zero cost, Strong community support	Steeper learning curve, Inconsistent hardware support, Fragmented distributions
macOS	Hybrid (XNU Kernel)	APFS	Virtual Memory + Compression	Pre-emptive Multitasking	System Integrity Protection, Gatekeeper, FileVault, TCC	Graphical (Aqua)	Creative professionals, Designers, Power users	Superior build quality & integration, Excellent security & privacy, Optimised performance	Expensive hardware, Closed ecosystem, Limited hardware choices
Android	Monolithic (Linux Kernel)	ext4 / F2FS	Virtual Memory + Low Memory Killer	Pre-emptive Multitasking	App Sandboxing, Runtime Permissions, Verified Boot	Graphical (Touch-based)	Mobile users, Developers, OEMs	Massive app ecosystem, High customisability,	Severe fragmentation, Variable security

								Broad device compatibility	quality, Battery & performance issues
iOS	Hybrid (XNU Kernel)	APFS	Virtual Memory + Jetsam	Pre-emptive Multitasking	Strict App Sandboxing, App Review Process, Secure Enclave	Graphical (Touch-based)	Mobile users seeking security & simplicity	Industry-leading security & privacy, Excellent optimisation, Long-term support	Highly restrictive ecosystem, Expensive devices, Limited user customisation

Critical Analysis and Key Insights

This activity highlighted to me several important distinctions between different operating systems. Linux seems to stand out for its open-source nature and flexibility, making it the preferred choice for servers, cloud infrastructure, and developers who require extremely granular control. Its monolithic kernel design allows for high performance but requires careful configuration to maintain stability across diverse hardware.

In contrast, Windows 11 prioritises compatibility and user accessibility. Its hybrid kernel and extensive driver support make it the dominant platform for desktop computing and gaming, which I daily-drive myself. However, this comes at the cost of increased complexity and a larger attack surface for hackers and malware, as evidenced by its reliance on regular security updates and third-party antivirus solutions.

Apple's macOS and iOS demonstrate the advantages of tight hardware and software integration. When Apple control the hardware offered in their machines, they are able to dial in their OS software precisely to minimise incompatibilities. The XNU kernel, combined with features such as System Integrity Protection and the Secure Enclave, delivers some of the strongest security and privacy protections available to consumers. I note that this however comes at the expense of user freedom and significantly higher costs when compared to Windows and Linux.

Android and iOS illustrate the fundamental differences between open and closed mobile ecosystems. While Android offers greater customisation, personalisation and device variety, iOS provides a more consistent and secure user experience due to Apple's strict control over both hardware and software, similar to their desktop computing OS.

Overall, this comparison reinforced the idea that there is no single "best" operating system. Instead, the choice of Operating System is driven by specific requirements such as cost, security needs, hardware constraints, and the technical expertise of the user.

Personal Reflection

Completing this activity significantly improved my understanding of why different operating systems exist and how their design choices directly impact performance, security, and usability. I'll find this knowledge valuable in my role as an IT Infrastructure Technician, where I will likely work with multiple platforms across different environments.

Appendix 2 – Lecture Activities / Activity Log

Although I was unable to attend the scheduled lectures in person, I completed self-directed study using the module handbook, recommended textbook (Operating System Concepts by Silberschatz et al.), and online resources to cover the key topics of process management, memory management, file systems, and scheduling. This independent study directly supported the content of this report.

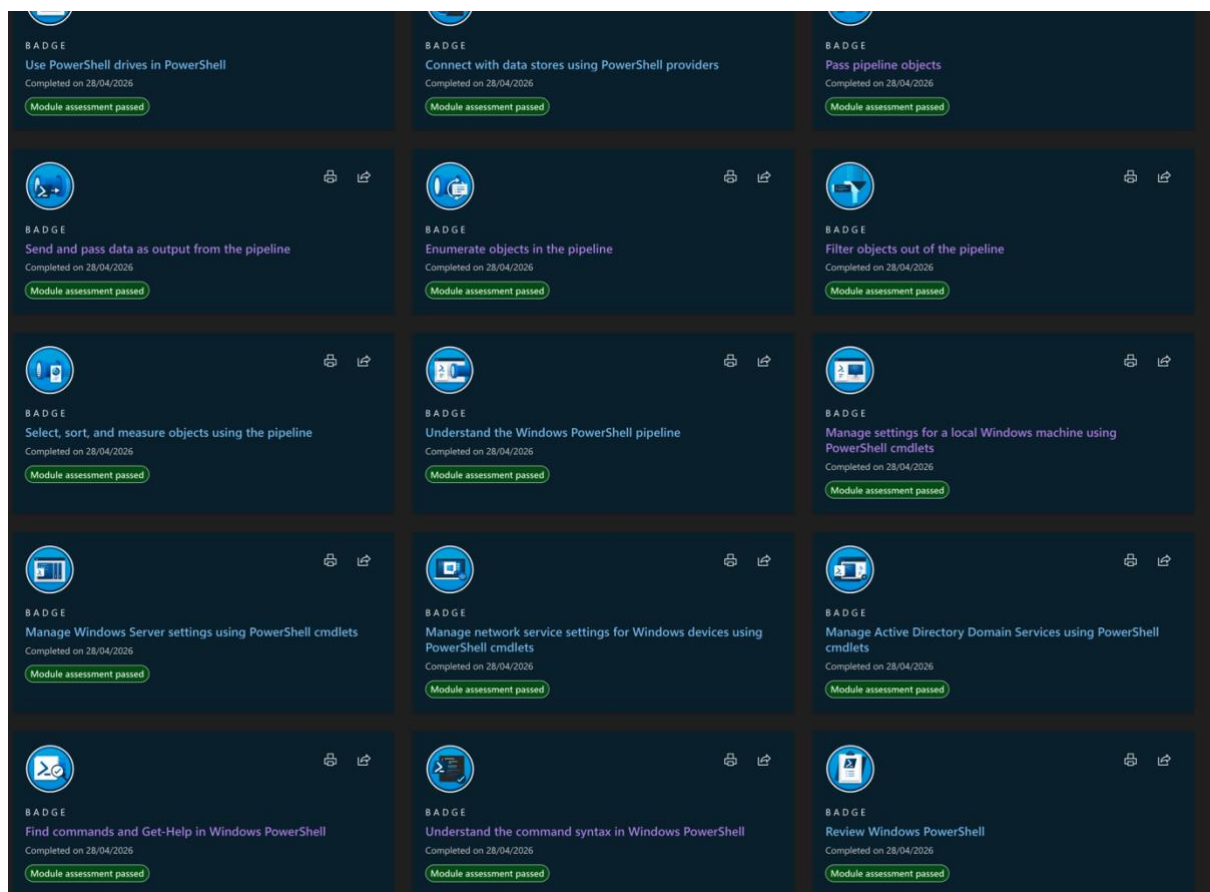
Skills I picked up during self-directed study activities included:

Date	Activity	Description	Skills Developed	Reflection
20–22 March 2026	Process States & Scheduling	Studied process lifecycle diagrams and compared FCFS, SJF and Round-Robin scheduling using online resources and module materials.	Process management, Algorithm evaluation	Helped me understand how the OS maintains system responsiveness.
25–27 March 2026	Memory Management	Worked through paging vs segmentation examples and calculated page table sizes using the module handbook.	Memory allocation techniques	Improved my understanding of virtual memory and fragmentation.
2–4 April 2026	File Systems	Compared NTFS, ext4 and APFS using the recommended textbook and online tutorials.	File system architecture	Gave me a clear understanding of why journaling is important for data integrity.
8–10 April 2026	Deadlock Prevention	Practised using Banker's Algorithm and resource allocation graphs from the module resources.	Deadlock theory, Critical analysis	One of the most useful topics — directly relevant to concurrent programming.

Appendix 3 – Evidence of Engagement with Online Certified Course

To further develop my practical skills in Windows command-line automation, I completed a significant portion of the Microsoft Learn learning path: AZ-040T00-A – Automate Administration with PowerShell.

This free Microsoft Learn path focuses on using PowerShell to automate administrative tasks on Windows systems. It directly supported the practical work I carried out in Task 4 of this assignment (Windows OS scripts).



The screenshot above shows multiple completed modules and badges earned on 28 April 2026, including:

- Use PowerShell drives in PowerShell
- Connect with data stores using PowerShell providers
- Send and pass data as output from the pipeline
- Enumerate objects in the pipeline
- Filter objects out of the pipeline
- Select, sort, and measure objects using the pipeline
- Understand the Windows PowerShell pipeline
- Manage settings for a local Windows machine using PowerShell cmdlets
- Manage Windows Server settings using PowerShell cmdlets
- Manage network service settings for Windows devices using PowerShell cmdlets
- Manage Active Directory Domain Services using PowerShell cmdlets
- Find commands and Get-Help in Windows PowerShell
- Understand the command syntax in Windows PowerShell
- Review Windows PowerShell