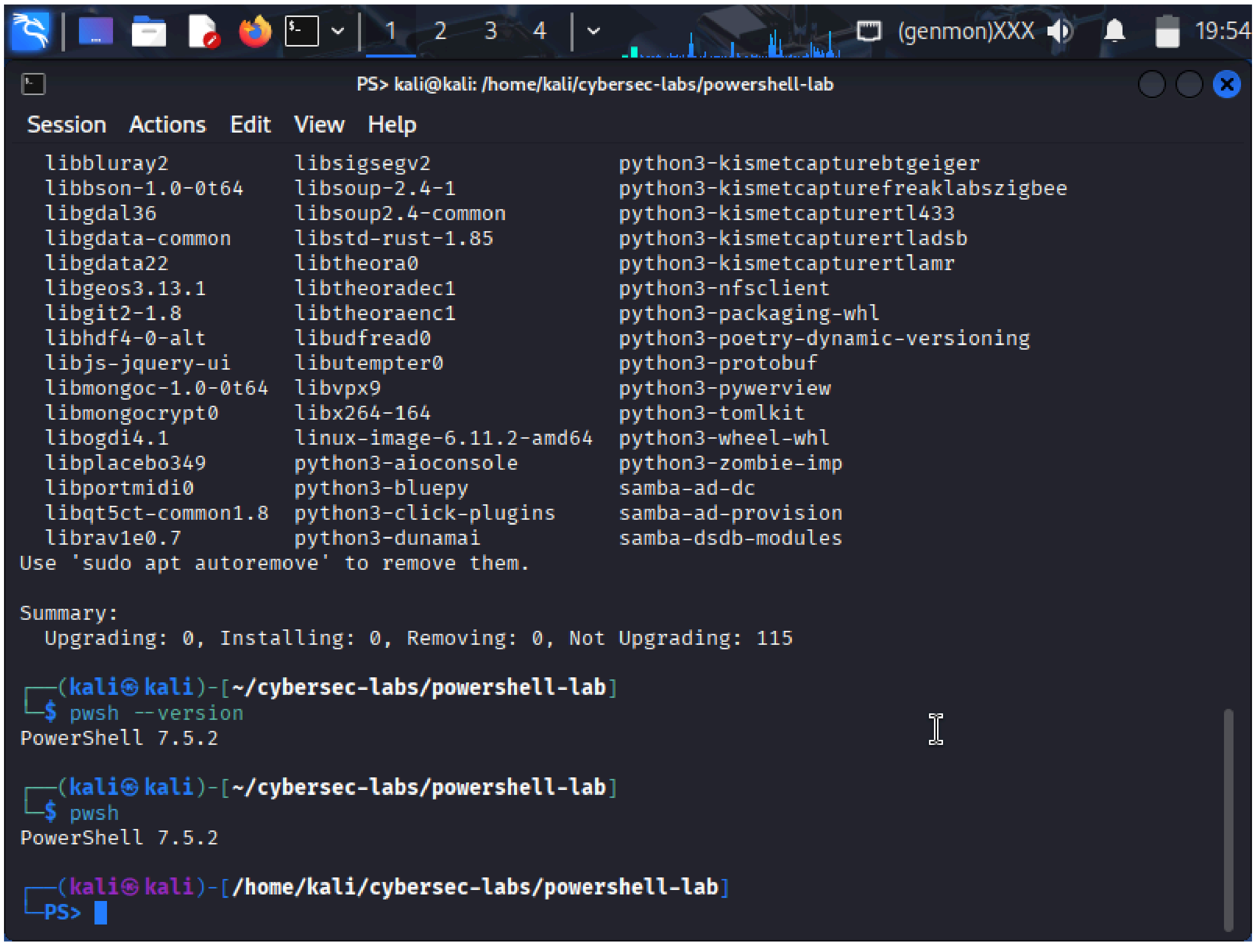
# Lab 7: Introduction to PowerShell

**Matthew Cox - COMP 325**

## Introduction:

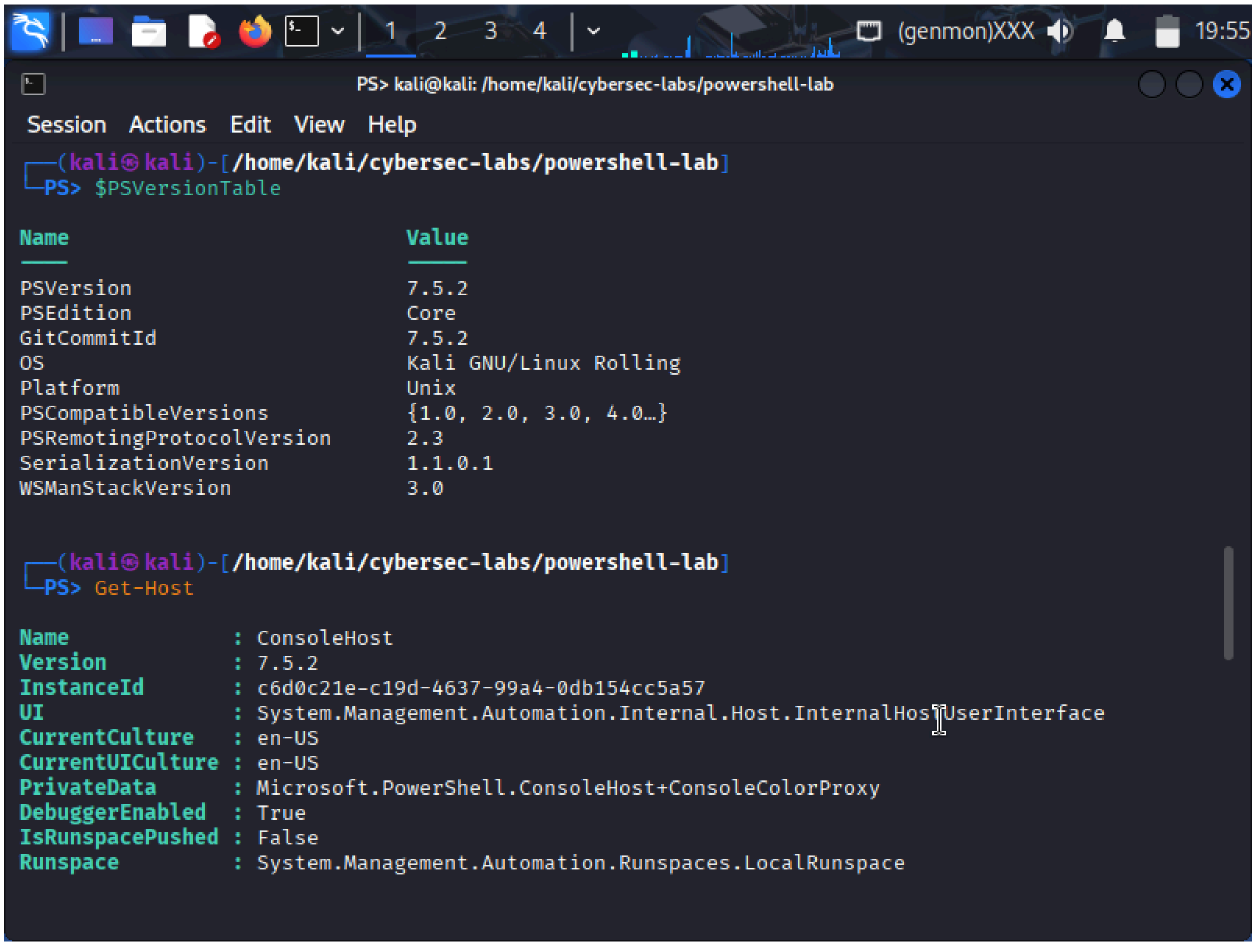
This lab provides a fundamental introduction to PowerShell, Microsoft's powerful, cross-platform command-line shell and scripting language, with a focus on its applications in cybersecurity. By completing this lab, I will learn to install and configure PowerShell on Linux, execute its basic commands, understand cmdlet syntax, and utilize it for system administration and security tasks. I will apply these skills to Windows security assessment, incident response, and be able to create simple PowerShell scripts relevant to cybersecurity operations, which is essential given the prevalence of Windows systems in enterprise environments.

## Body:


1 2 3 4
(genmon)XXX
19:54
PS> kali@kali: /home/kali/cybersec-labs/powershell-lab
Session Actions Edit View Help

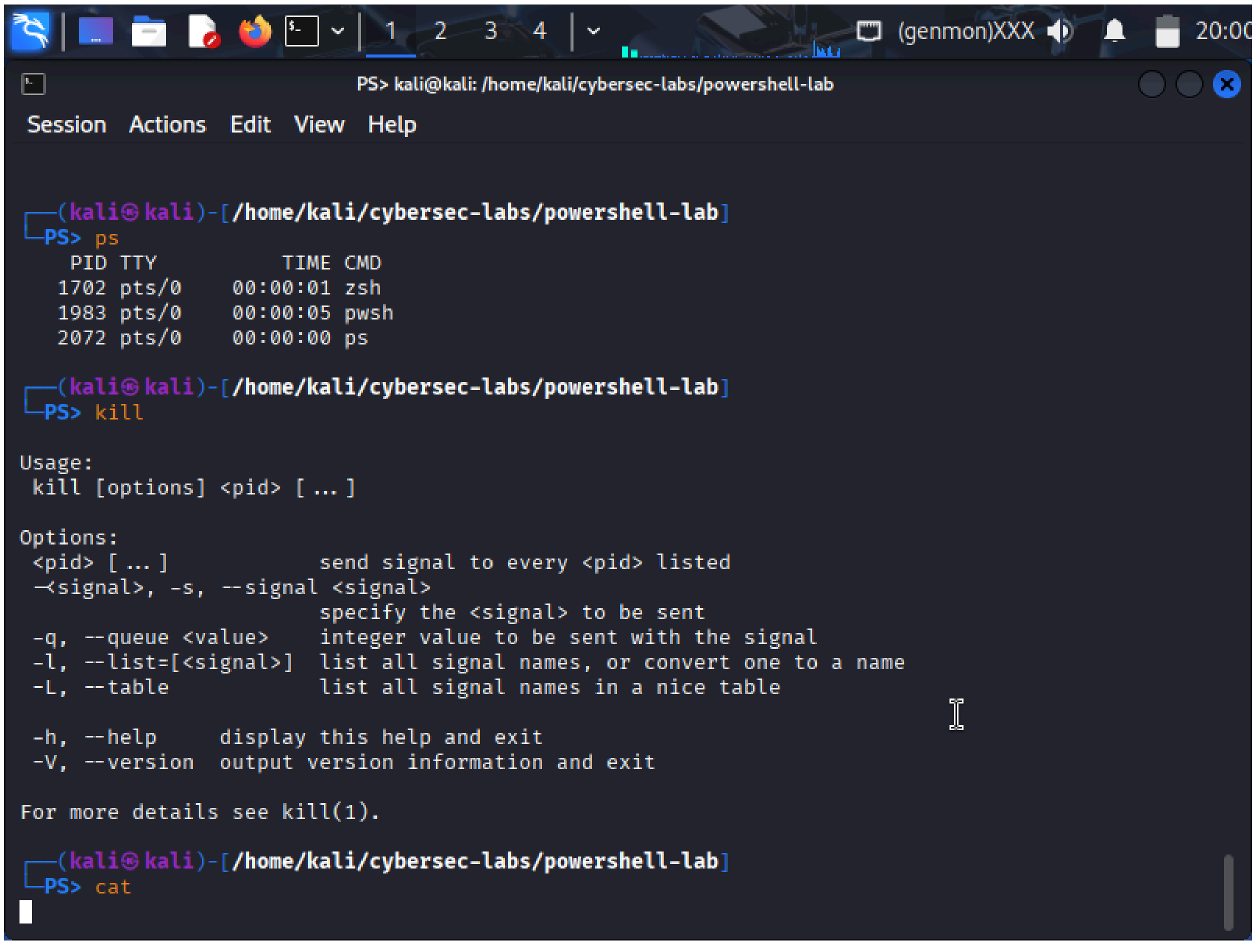
```
  libbluray2          libsigsegv2                python3-kismetcapturebtgeiger
  libbson-1.0-0t64    libsoup-2.4-1              python3-kismetcapturefreaklabszigbee
  libgdal36           libsoup2.4-common          python3-kismetcapturertl433
  libgdata-common     libstd-rust-1.85           python3-kismetcapturertladsb
  libgdata22          libtheora0                 python3-kismetcapturertlamr
  libgeos3.13.1       libtheoradec1              python3-nfsclient
  libgit2-1.8         libtheoraenc1              python3-packaging-whl
  libhdf4-0-alt       libudfread0                python3-poetry-dynamic-versioning
  libjs-jquery-ui     libutempter0               python3-protobuf
  libmongoc-1.0-0t64  libvpx9                    python3-pywerview
  libmongocrypt0      libx264-164                python3-tomlkit
  libogdi4.1          linux-image-6.11.2-amd64   python3-wheel-whl
  libplacebo349       python3-aioconsole         python3-zombie-imp
  libportmidi0        python3-bluepy             samba-ad-dc
  libqt5ct-common1.8  python3-click-plugins      samba-ad-provision
  librav1e0.7         python3-dunamai            samba-dsdb-modules
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 115

┌──(kali㉿kali)-[~/cybersec-labs/powershell-lab]
└─$ pwsh --version
PowerShell 7.5.2

┌──(kali㉿kali)-[~/cybersec-labs/powershell-lab]
└─$ pwsh
PowerShell 7.5.2

┌──(kali㉿kali)-[/home/kali/cybersec-labs/powershell-lab]
└─PS> 
```

While I was setting up this environment, I ran into a few issues. At first I wasnt using pfsense when using my kali linux. Obviously, for this lab needing a connection for update and upgrade, not having a connection was going to leave me at a dead end. But, I seemed to run into the same issue even when I booted up pfsense. I then ran my machine on a NAT Network. It works, but I am still unsure why. Now, when I use kali linux, it does bug out and or crash the first time loading it up. After that first misfire, everything seems to run smoothly. In the image above, you can see I made it through the update and upgrade and was able to use powershell.



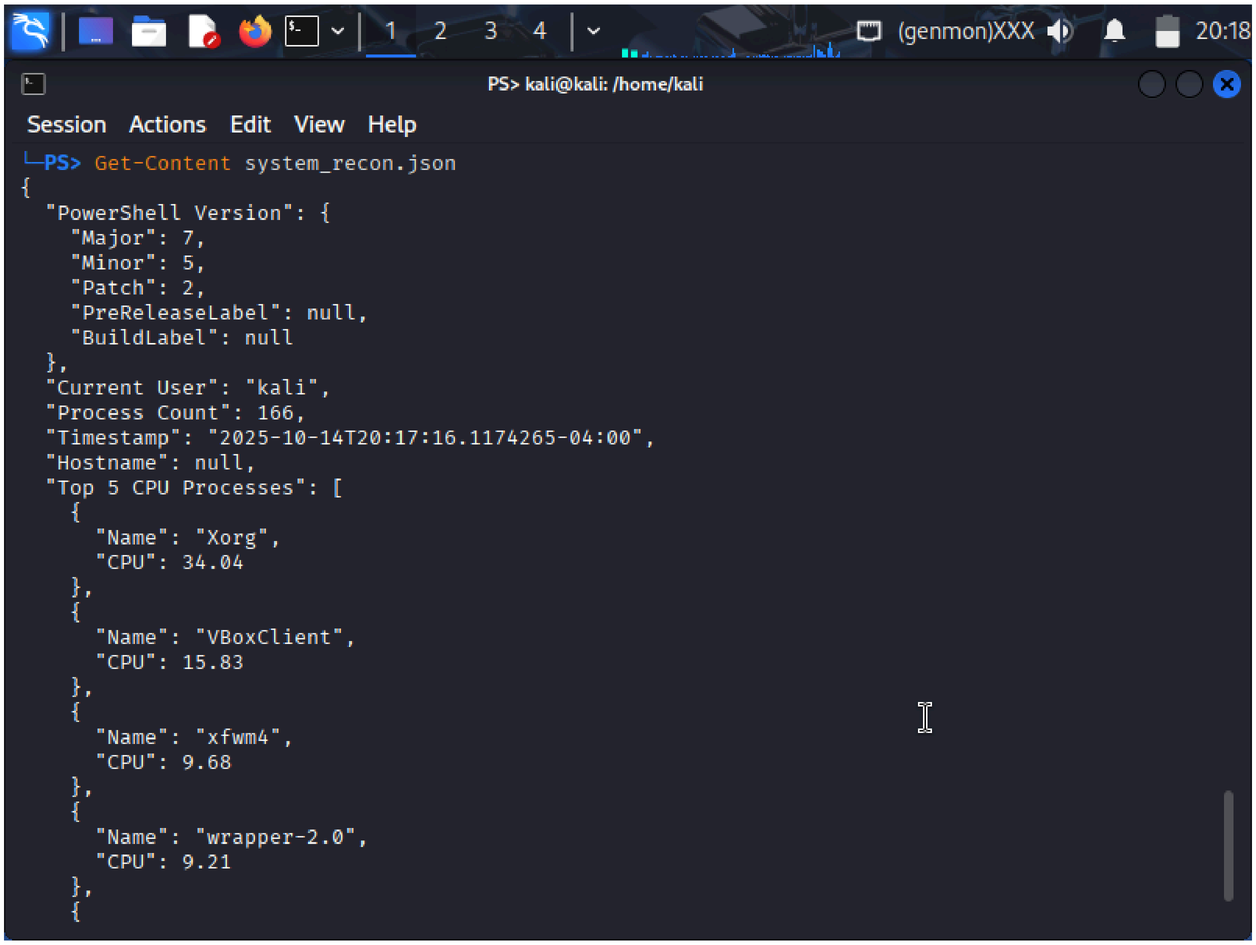
```
┌──(kali㉿kali)-[/home/kali/cybersec-labs/powershell-lab]
└─PS> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      7.5.2
PSEdition                      Core
GitCommitId                    7.5.2
OS                             Kali GNU/Linux Rolling
Platform                       Unix
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0…}
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
WSManStackVersion              3.0


┌──(kali㉿kali)-[/home/kali/cybersec-labs/powershell-lab]
└─PS> Get-Host

Name             : ConsoleHost
Version          : 7.5.2
InstanceId       : c6d0c21e-c19d-4637-99a4-0db154cc5a57
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : en-US
CurrentUICulture : en-US
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled  : True
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
```

In this image, I am just using some basic Powershell commands to get familiar with the environment.


(genmon)XXX
20:00
PS> kali@kali: /home/kali/cybersec-labs/powershell-lab
Session Actions Edit View Help

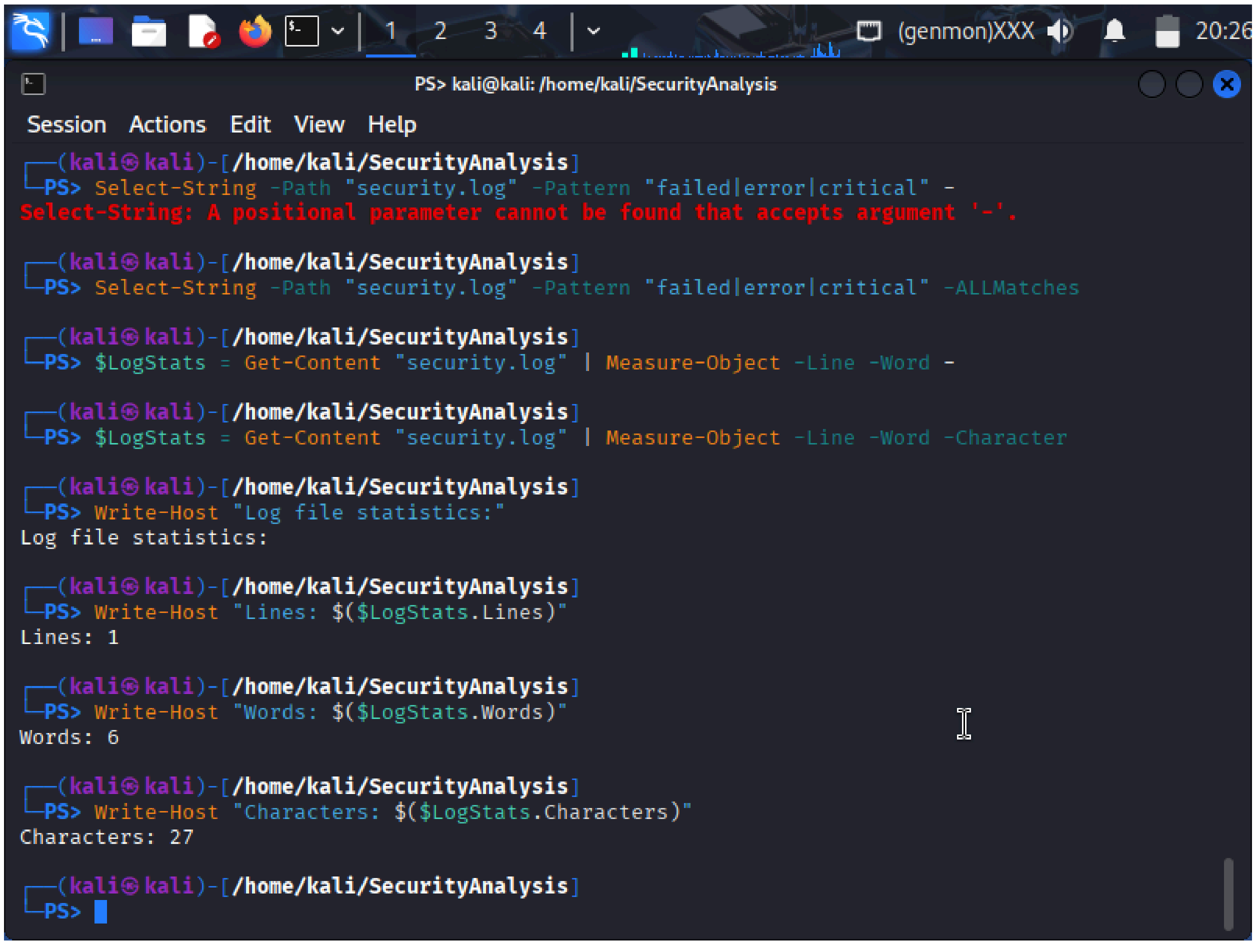
```
┌──(kali㉿kali)-[/home/kali/cybersec-labs/powershell-lab]
└─PS> ps
    PID TTY          TIME CMD
   1702 pts/0    00:00:01 zsh
   1983 pts/0    00:00:05 pwsh
   2072 pts/0    00:00:00 ps

┌──(kali㉿kali)-[/home/kali/cybersec-labs/powershell-lab]
└─PS> kill

Usage:
 kill [options] <pid> [...]

Options:
 <pid> [...]            send signal to every <pid> listed
 -<signal>, -s, --signal <signal>
                        specify the <signal> to be sent
 -q, --queue <value>    integer value to be sent with the signal
 -l, --list=[<signal>]  list all signal names, or convert one to a name
 -L, --table            list all signal names in a nice table

 -h, --help     display this help and exit
 -V, --version  output version information and exit

For more details see kill(1).

┌──(kali㉿kali)-[/home/kali/cybersec-labs/powershell-lab]
└─PS> cat
```

Here, I am using a list of shortcuts provided in Powershell. Again, just getting familiar with the environment.



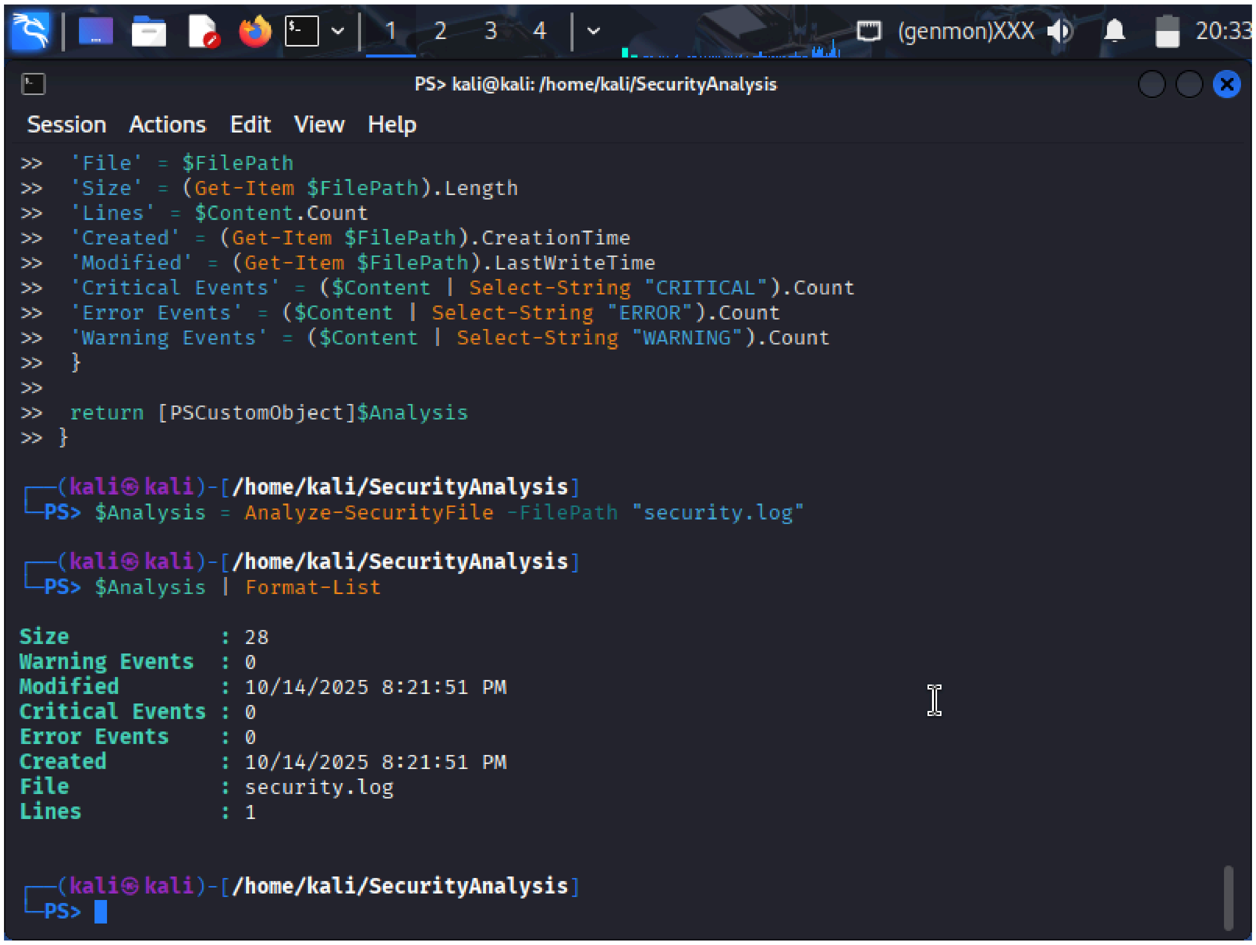
```
└─PS> Get-Content system_recon.json
{
  "PowerShell Version": {
    "Major": 7,
    "Minor": 5,
    "Patch": 2,
    "PreReleaseLabel": null,
    "BuildLabel": null
  },
  "Current User": "kali",
  "Process Count": 166,
  "Timestamp": "2025-10-14T20:17:16.1174265-04:00",
  "Hostname": null,
  "Top 5 CPU Processes": [
    {
      "Name": "Xorg",
      "CPU": 34.04
    },
    {
      "Name": "VBoxClient",
      "CPU": 15.83
    },
    {
      "Name": "xfwm4",
      "CPU": 9.68
    },
    {
      "Name": "wrapper-2.0",
      "CPU": 9.21
    },
    {
```

In this image, I was able to successfully create a system reconnaissance script.


(genmon)XXX
20:26
PS> kali@kali: /home/kali/SecurityAnalysis
Session Actions Edit View Help

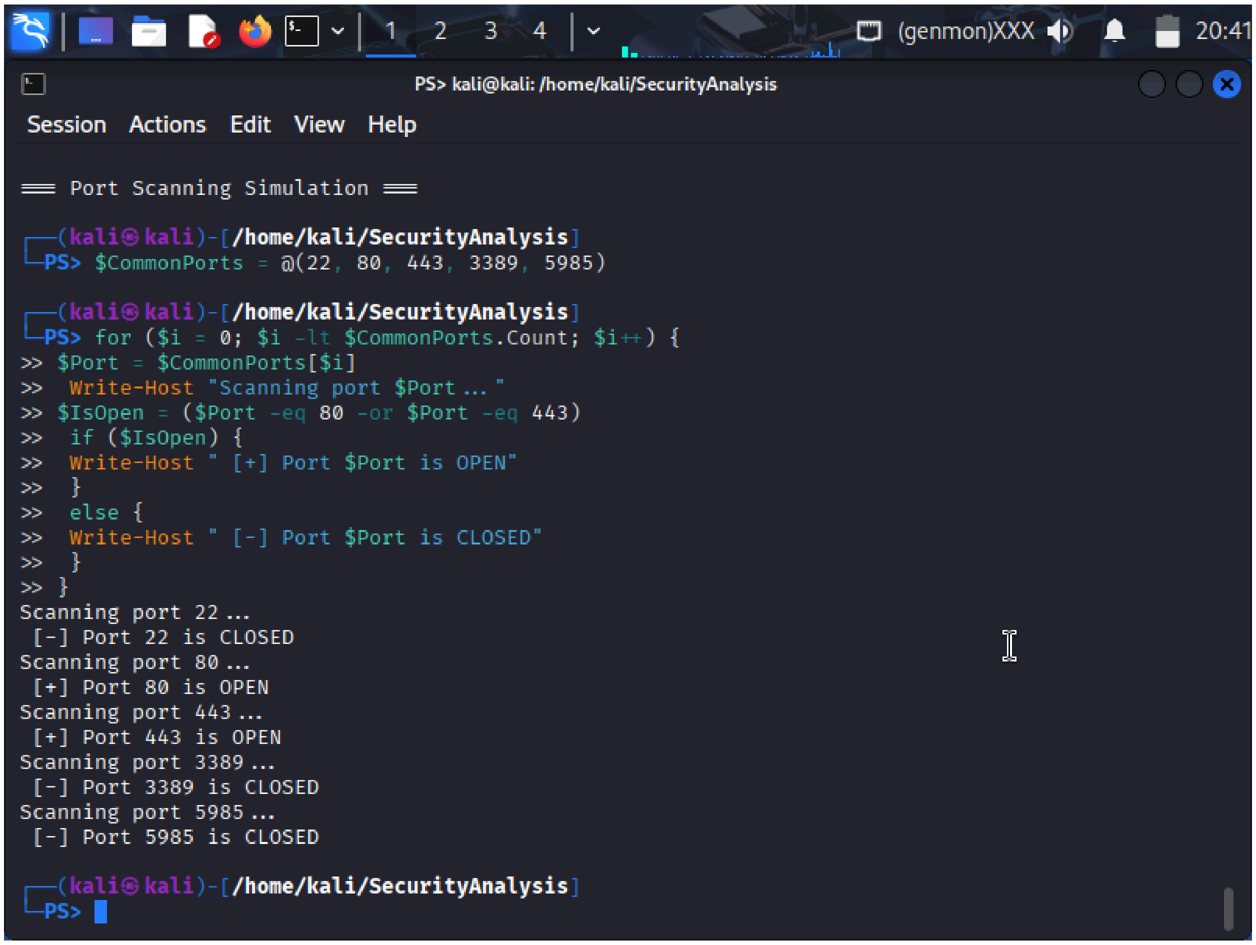
```
┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> Select-String -Path "security.log" -Pattern "failed|error|critical" -
Select-String: A positional parameter cannot be found that accepts argument '-'.

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> Select-String -Path "security.log" -Pattern "failed|error|critical" -ALLMatches

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> $LogStats = Get-Content "security.log" | Measure-Object -Line -Word -

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> $LogStats = Get-Content "security.log" | Measure-Object -Line -Word -Character

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> Write-Host "Log file statistics:"
Log file statistics:

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> Write-Host "Lines: $($LogStats.Lines)"
Lines: 1

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> Write-Host "Words: $($LogStats.Words)"
Words: 6

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> Write-Host "Characters: $($LogStats.Characters)"
Characters: 27

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS>
```

In this image, I was doing some file searching and filtering. Above the file searching and filtering, you can see I ran into my fair share of obstacles. Powershell is definitely a work in progress. I can't say I'm a huge fan, but it is challenging and I'm learning along the way.


(genmon)XXX
20:33
PS> kali@kali: /home/kali/SecurityAnalysis
Session Actions Edit View Help

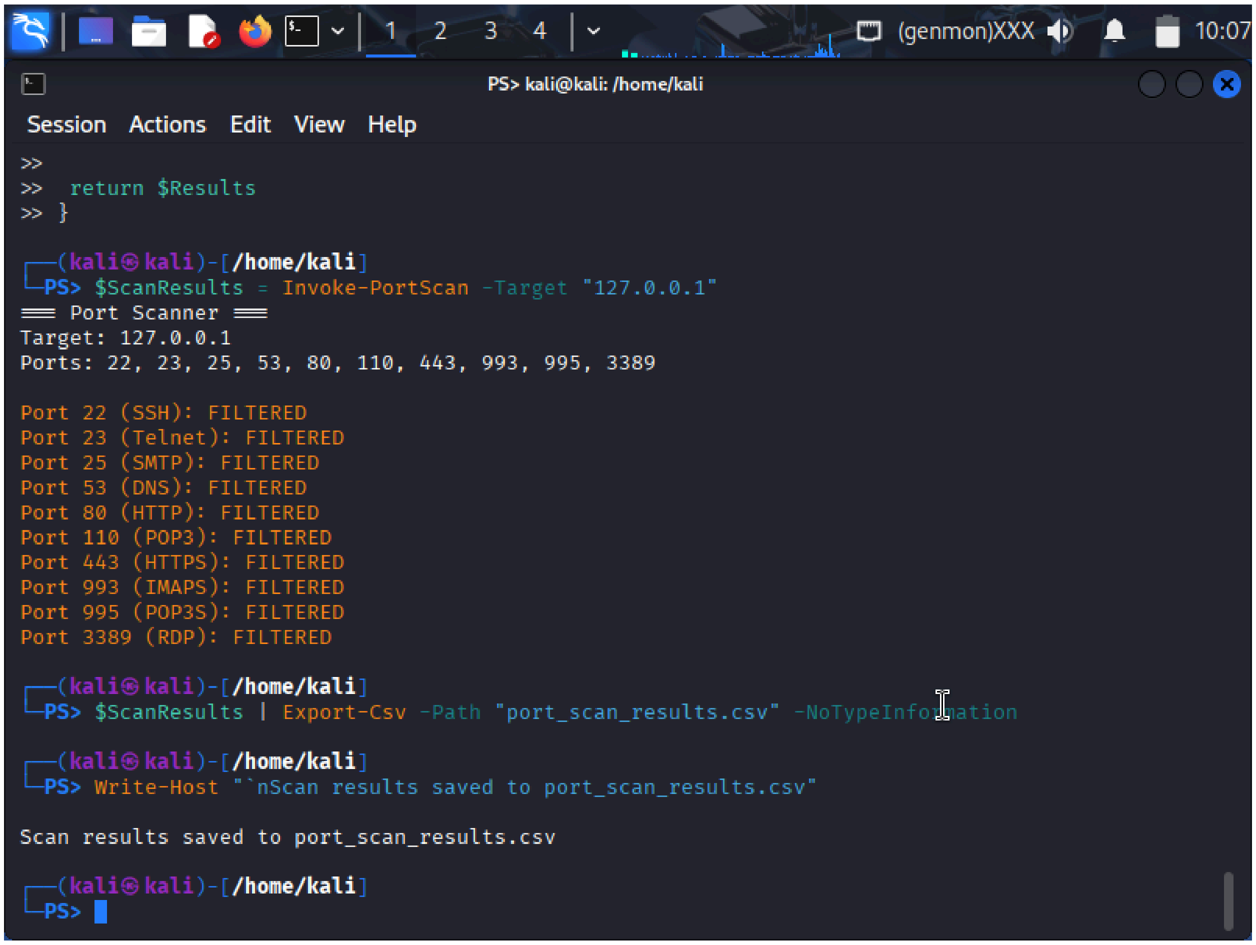
```
>>  'File' = $FilePath
>>  'Size' = (Get-Item $FilePath).Length
>>  'Lines' = $Content.Count
>>  'Created' = (Get-Item $FilePath).CreationTime
>>  'Modified' = (Get-Item $FilePath).LastWriteTime
>>  'Critical Events' = ($Content | Select-String "CRITICAL").Count
>>  'Error Events' = ($Content | Select-String "ERROR").Count
>>  'Warning Events' = ($Content | Select-String "WARNING").Count
>>  }
>>
>>  return [PSCustomObject]$Analysis
>> }

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> $Analysis = Analyze-SecurityFile -FilePath "security.log"

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> $Analysis | Format-List

Size            : 28
Warning Events  : 0
Modified        : 10/14/2025 8:21:51 PM
Critical Events : 0
Error Events    : 0
Created         : 10/14/2025 8:21:51 PM
File            : security.log
Lines           : 1


┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS>
```

In this image, I committed some advanced file operations. No issues here.


1 2 3 4
(genmon)XXX
20:41
PS> kali@kali: /home/kali/SecurityAnalysis
Session Actions Edit View Help

```
══ Port Scanning Simulation ══

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> $CommonPorts = @(22, 80, 443, 3389, 5985)

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS> for ($i = 0; $i -lt $CommonPorts.Count; $i++) {
>> $Port = $CommonPorts[$i]
>>  Write-Host "Scanning port $Port..."
>> $IsOpen = ($Port -eq 80 -or $Port -eq 443)
>>  if ($IsOpen) {
>>  Write-Host " [+] Port $Port is OPEN"
>>  }
>>  else {
>>  Write-Host " [-] Port $Port is CLOSED"
>>  }
>> }
Scanning port 22...
 [-] Port 22 is CLOSED
Scanning port 80...
 [+] Port 80 is OPEN
Scanning port 443...
 [+] Port 443 is OPEN
Scanning port 3389...
 [-] Port 3389 is CLOSED
Scanning port 5985...
 [-] Port 5985 is CLOSED

┌──(kali㉿kali)-[/home/kali/SecurityAnalysis]
└─PS>
```

In this exercise, it was a focus on Powershell scripting fundamentals. Which included conditional statements and loops for automation.



```
>>
>>  return $Results
>> }

┌──(kali㉿kali)-[/home/kali]
└PS> $ScanResults = Invoke-PortScan -Target "127.0.0.1"
══ Port Scanner ══
Target: 127.0.0.1
Ports: 22, 23, 25, 53, 80, 110, 443, 993, 995, 3389

Port 22 (SSH): FILTERED
Port 23 (Telnet): FILTERED
Port 25 (SMTP): FILTERED
Port 53 (DNS): FILTERED
Port 80 (HTTP): FILTERED
Port 110 (POP3): FILTERED
Port 443 (HTTPS): FILTERED
Port 993 (IMAPS): FILTERED
Port 995 (POP3S): FILTERED
Port 3389 (RDP): FILTERED

┌──(kali㉿kali)-[/home/kali]
└PS> $ScanResults | Export-Csv -Path "port_scan_results.csv" -NoTypeInformation

┌──(kali㉿kali)-[/home/kali]
└PS> Write-Host "`nScan results saved to port_scan_results.csv"

Scan results saved to port_scan_results.csv

┌──(kali㉿kali)-[/home/kali]
└PS>
```

In this image you can see I developed a hash calculation utility. Specifically, a hash calculator for forensics

**Knowledge Assesment:**

**Question 1:** True or False: PowerShell uses a Verb-Noun syntax for its cmdlets (e.g., Get-Process, SetLocation).
**True**

**Question 2**: Which PowerShell cmdlet would you use to get detailed information about running processes?
**b) Get-Process**

**Question 3**: True or False: PowerShell treats everything as objects rather than plain text.
**True**

**Question 4**: What does the PowerShell pipeline operator | accomplish?
**b) Passes objects from one cmdlet to another**

**Question 5**: Which command would filter PowerShell output to show only processes using more than 100MB of memory?
**a) Get-Process | Where-Object {$_.WorkingSet -gt 100MB}**

**Conclusion**:
PowerShell isn't just a Windows system administration tool, it is an essential, cross-platform skill for any modern cybersecurity professional. Whether you are performing Windows Security Assessments, executing Incident Response procedures, managing Active Directory Security, or developing Automation scripts, proficiency in PowerShell is non-negotiable. Furthermore, its pervasive use in Post-Exploitation scenarios means defenders must master it to effectively detect and counteract advanced threats. Having completed this lab, I can now execute commands and understand cmdlet syntax, use the tool for system information gathering and analysis, and apply my scripting skills to complex security and Compliance Auditing scenarios.