PowerShell

Shell Techniques, Tools, and Script Building Blocks

Windows Server

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Have a Question?

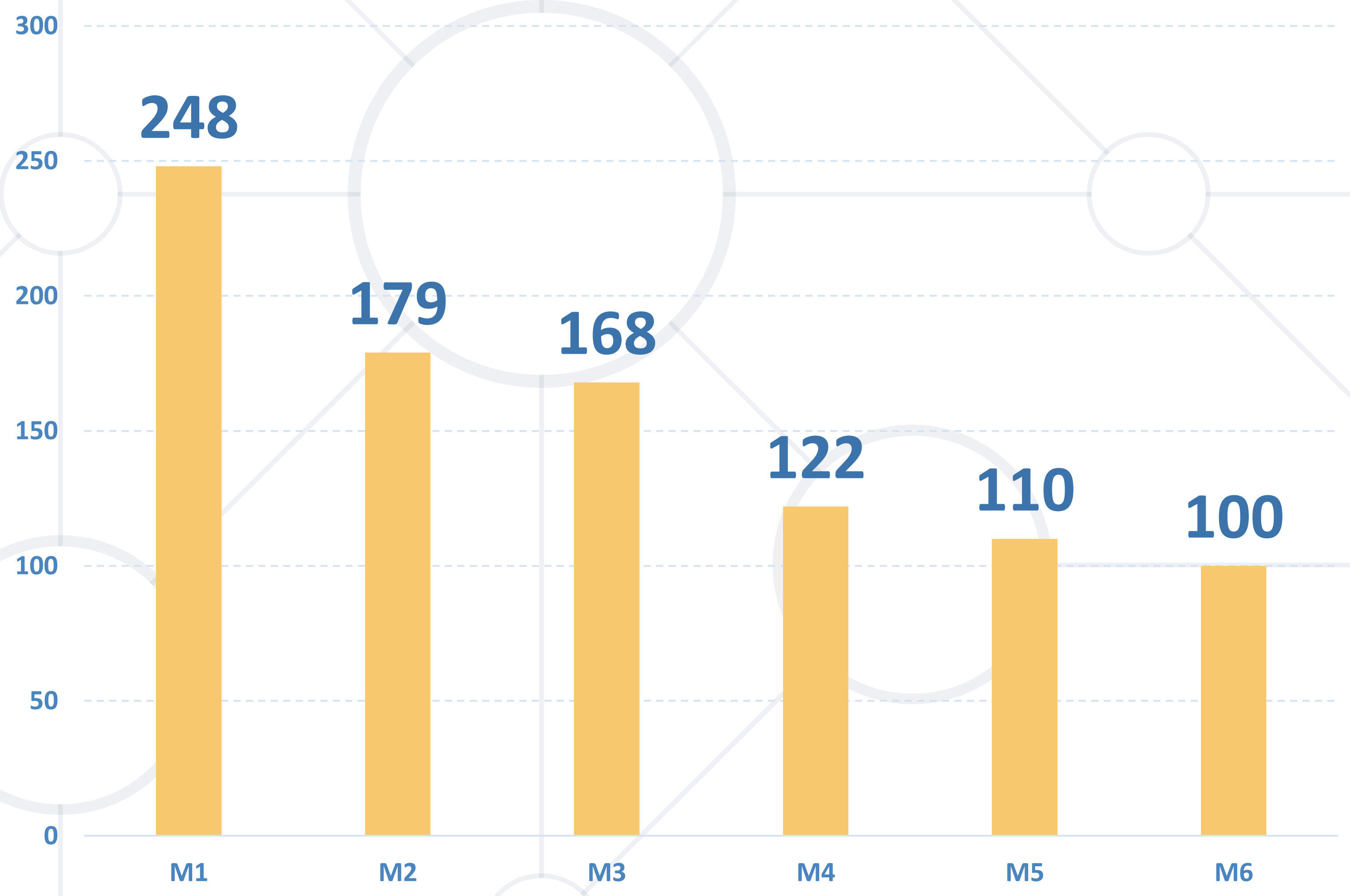
sli.do

#WSA

facebook.com/groups/

WindowsSystemAdministrationMarch2023/

Homework Progress



Submit M6
until 23:59:59
on 12.05.2023

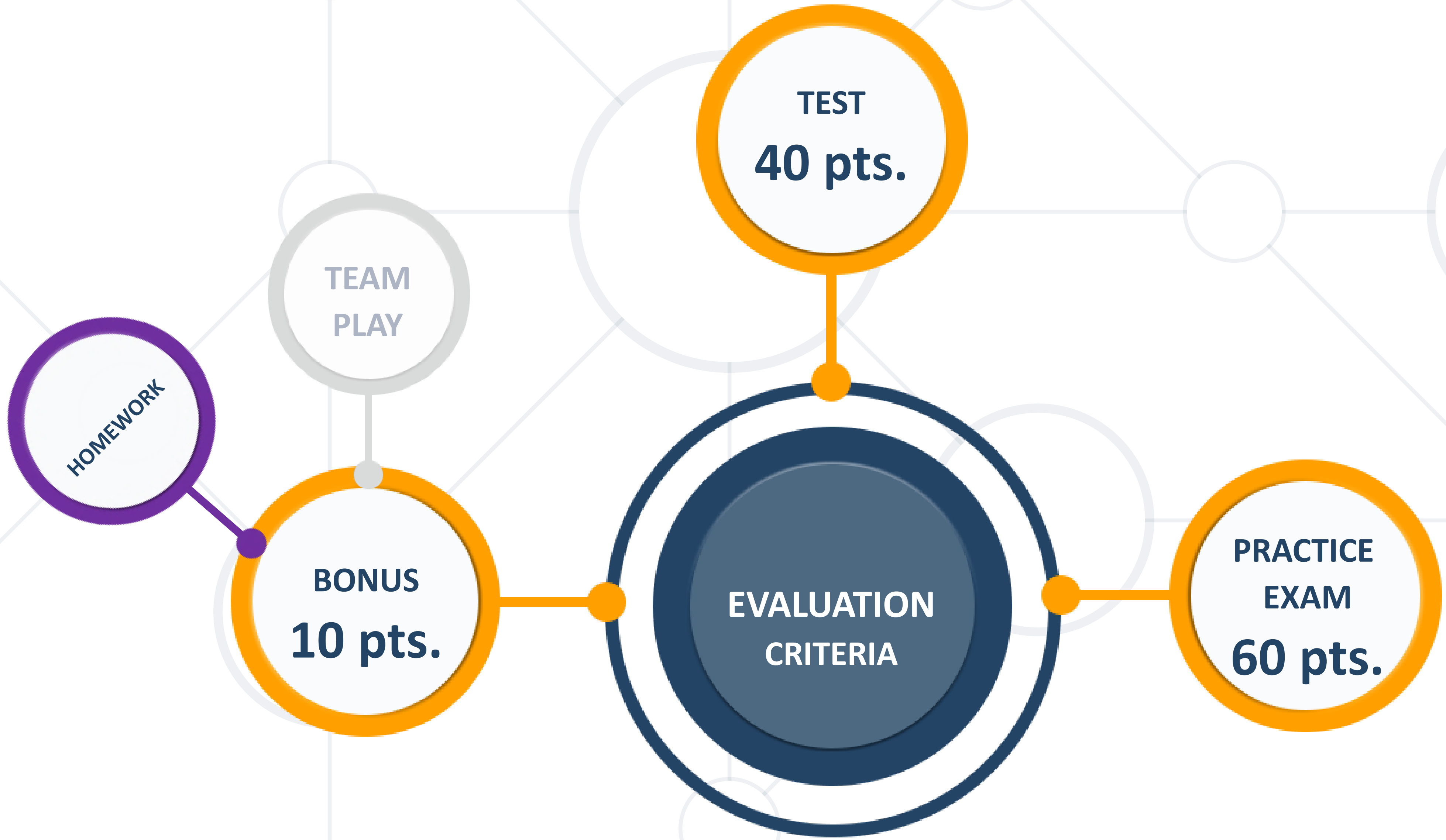
Submit M7
until 23:59:59
on 19.05.2023

The End is Near ☺

THIS MODULE

AND ONE MORE TO GO

Scoring for Windows System Administration



Practice (Remote)

- (Install and) configure a few machines
- Connect them in a certain way
- Install and configure a set of services
- Create a set of users and groups
- Set the appropriate permissions
- Write a simple script that does what requested
- Additional tasks as per the exam requirements

The **practice exam** will be held **remotely** in a **controlled environment**

All you need is just a PC with **RDP client** and **Internet connectivity**

You will have **4 hours**

Test (Remote)

Software University

30

minutes

20

single-choice questions

10

multi-choice questions

Test Your Knowledge

Practice (exam-like) questions:

https://zahariev.pro/q/wsa

Previous Week (M6)

Quick overview

What We Covered

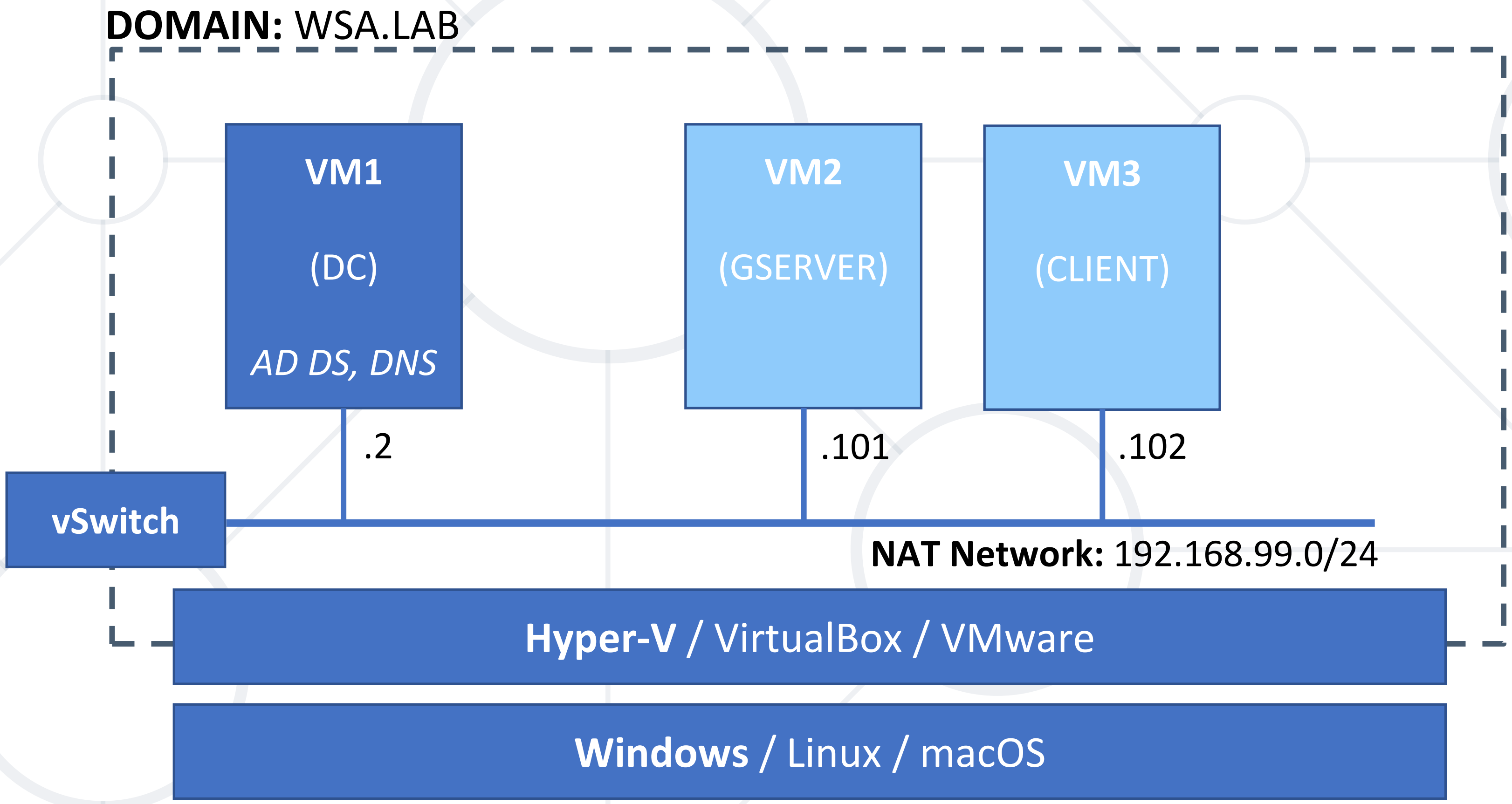
- Troubleshooting
- Monitoring
- Backup and Restore
- Scheduling

This Module (M7)

Topics

Table of Contents

Lab Infrastructure


DOMAIN: WSA.LAB
VM1
(DC)
AD DS, DNS
VM2
(GSERVER)
VM3
(CLIENT)
.2
.101
.102
vSwitch
NAT Network: 192.168.99.0/24
Hyper-V / VirtualBox / VMware
Windows / Linux / macOS

Work in the Shell

Know Your Console

Getting Help

- Our indispensable source is **Get-Help**
- Many modifiers and one to combine them **-ShowWindow**
- We can use **Get-Member** to examine an object
- Or **Get-Command -Noun Topic**, to get list of related commands
- If wondering what actions are there, then use **Get-Verb**
- When installing new modules always use **Update-Help**
- Of course, we can **use on-line** sources, but we should be **careful**

Aliases

- Instead of typing **long commands** we can **shorten** them
- There are some pre-built aliases - **Get-Alias** or just **alias**
- We can create our own with **New-Alias**
- Or we can alter existing one with **Set-Alias**
- Load or save aliases with **Import-Alias** and **Export-Alias**
- Unless actions are taken, they **live only** in **our session**
- We can sore them in our profile folder **$Home\Documents**
- Or system-wide in **$PsHome**

Common Commands

- Special **system aliases** make our life easier
- They mimic common **CMD.exe** and **UNIX** shell **commands**
- Among them we can find
  - **dir** and **ls** to examine folder's content
  - **cd**, **chdir**, **md**, **mkdir**, **rd**, **rmdir** to change, add, or remove folder
  - **cat** and **type** to explore file's content
  - **ps** to get list of running process and others

Environment Variables (CMD Shell)

- Before PowerShell there was **CMD Shell** and it is still there
- There are plenty of **existing** and **well-known** commands there
- We can examine its environment by executing **set**
- Variables are referenced by **%VARIABLE%**
- Include **%PATH%**, **%COMPUTERNAME%**, **%USERNAME%**, etc.
- Their value can be seen with **echo**

Environment Variables (PowerShell)

- **PowerShell** is de-facto a standard for the last several years
- Its environment can be explored with **Get-Variable**
- Of course, we can define our own variables with **New-Variable**
- Alter (**Set-Variable**) or clear value (**Clear-Variable**) of existing
- And finally dispose variables with **Remove-Variable**
- We can use PowerShell variables by name **$Variable**
- We can use CMD variables with **$env:VARIABLE**

Redirect Output

- Some commands have **internal export** functions
- Alternative option is the **output redirection**
- This one **history > C:\Temp\history.txt** will save session history
- Of course, we can use special commands like
  - **Out-File** or **Export-CSV**
- And we can export to a visual list **Out-GridView** or **ogv**
- Screen and file at the same time - **Tee-Object** or just **tee**

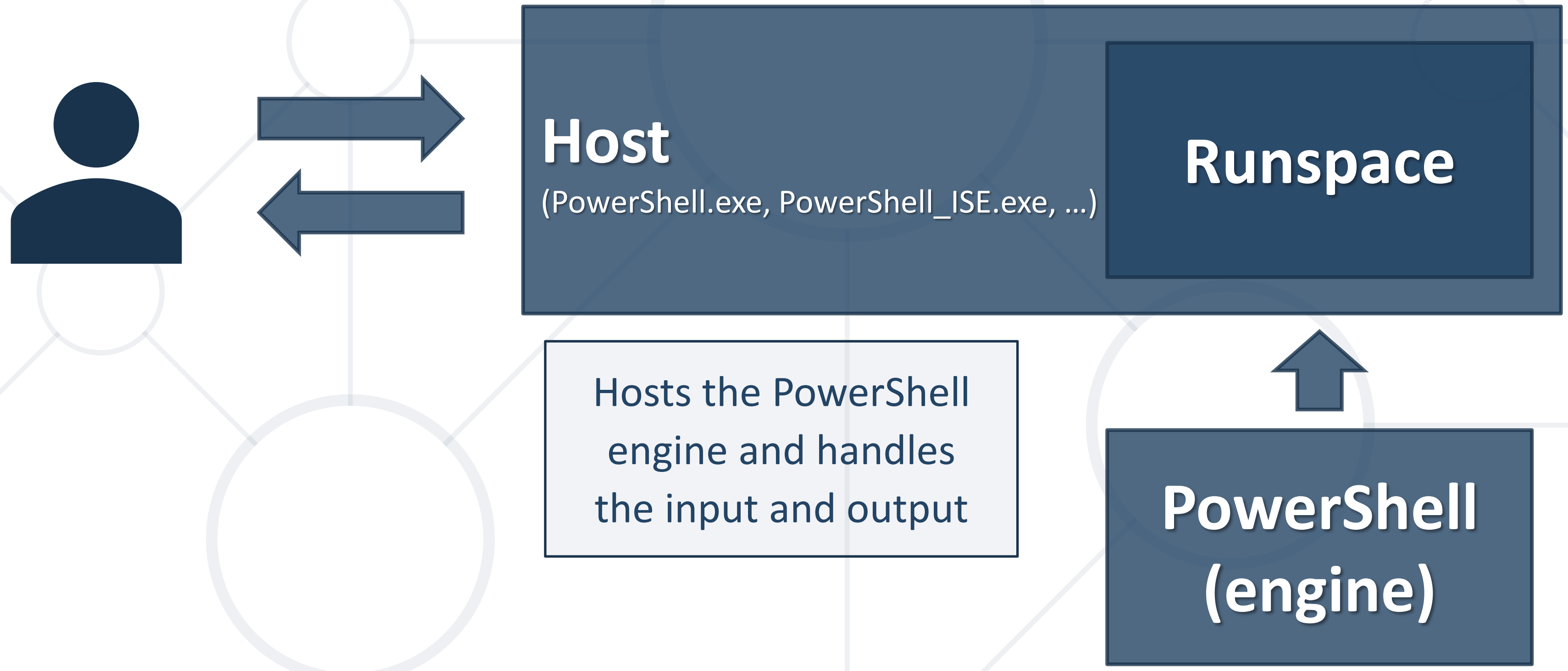
Files and Folders

- We can create new files or folders with **New-Item**
- Additionally, we can **copy**, **move**, **rename**, and **delete** items
- **Clear-Content** clears contents of a file
- **Get-Item** returns the item itself
- And **Get-ChildItems** returns its children (level 1)
- We can even add **-Recurse** to get other levels as well

PowerShell Profiles

- Added aliases, functions, and variables are not persistent
- To retain changes, we must put them in our profile
- Profile for **all users** and **all shells** (hosts)
  - **%windir%\system32\WindowsPowerShell\v1.0\profile.ps1**
- Profile for **all users**, but only to the **Microsoft.PowerShell shell**
  - **%windir%\system32\WindowsPowerShell\v1.0\Microsoft.PowerShell_profile.ps1**
- Profile for the **current user**, but affects **all shells**
  - **%UserProfile%\My Documents\WindowsPowerShell\profile.ps1**
- Profile for the **current user** and the **Microsoft.PowerShell shell**
  - **%UserProfile%\MyDocuments\WindowsPowerShell\Microsoft.PowerShell_profile.ps1**

(Power)Shell vs Host



Hosts the PowerShell engine and handles the input and output

An instance of the PowerShell engine

A set of .NET classes stored in DLL

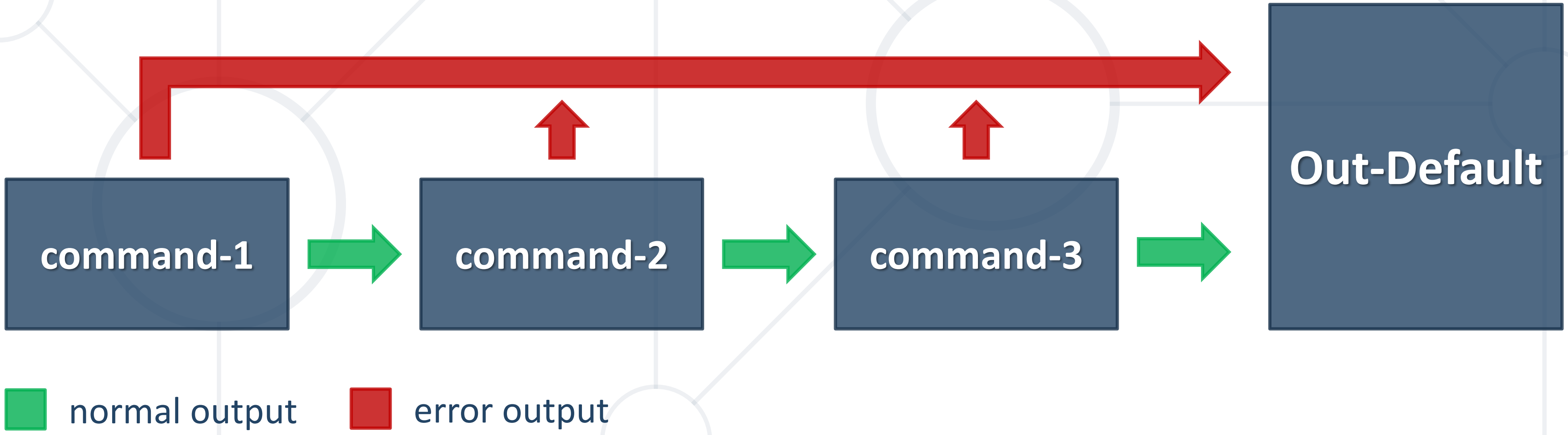
Command Sequences

- Series of **independent commands**

    command-1 ; command-2 ; command-3 ; …

- Series of **connected commands**

    command-1 | command-2 | command-3 | …



Practice: Work in the Shell

Live Demonstration in Class

Tools and Building Blocks

PowerShell Script Building Blocks

Comments

Local Variable

Flow Control

Command

```
#
# colors.ps1
#

$color = "blue"

if ($color -Eq "red") {
  write "red is the color"
} else {
  write "$color is not red"
}
```

Comments

- Single line

```
# One line comment
...
# and yet another one
```

- Multiline

```
# First line of short multiline comment
# Second line of short multiline comment
...
<#
  1-st line of long multiline comment
  ...
  N-th line of long multiline comment
#>
```

Variables and Data Types

- Variable is a **named storage** for information
- Some of the supported common data types
  - **[datetime]** - date or time
  - **[string]** - string of characters
  - **[int]** - 32-bit integer
  - **[double]** - double-precision floating number

```
[int]$choice = 5
```

- If not specified, a variable can **change** its **type on-the-fly**

Flow Control

- If-Then-Else structure

```
$limit = 50
$speed = 110
if ($speed -GT $limit) { "You are driving too fast" }
else { "You are a good driver" }
```

- Switch structure

```
$color = 2

switch ($color) {
  1 {"The color is red"}
  2 {"The color is blue"}
  default {"The color is black"}
}
```

```
GT (greater than)
GE (greater than or equal)
LT (less than)
LE (less than or equal)
EQ (equal)
NE (not equal)
LIKE (match wildcard)
NOTLIKE (does not match wildcard)
```

Repetitive Tasks

- For (repeat while the condition is true)

```
for ($i = 1; $i -LE 5; $i++) { ... }
```

- While (repeat while the condition is true)

```
$i = 1
while ($i -LE 5) { ... $i++ }
```

- Do While (repeat while the condition is true)

```
$i = 1
do { ... $i++ } while ($i -LE 5)
```

- Do Until (repeat while the condition is NOT true)

```
$i = 1
do { ... $i++ } until ($i -GT 5)
```

User Interaction

- Get user input

```
$choice = Read-Host -Prompt "Would you like to continue (Y/N)?"
```

- Write information back

```
Write-Host "Your choice is $choice"
```

- Write conditionally

```
Write-Verbose "### Here we do this and that"
...
PS C:\> ./script.ps1 -Verbose
...
### Here we do this and that
...
```

PowerShell ISE

- Powerful development environment
- Productivity improvements
  - Section **outlining** for readability
  - **Breakpoints** for debugging
- Available only for **Desktop Experience** enabled installations
- Can work simultaneously with **multiple** source **files**
- Supports **more** than one **PowerShell session** local or remote

https://docs.microsoft.com/en-us/powershell/scripting/core-powershell/ise/keyboard-shortcuts-for-the-windows-powershell-ise?view=powershell-5.1

Practice: Tools and Building Blocks

Live Demonstration in Class

Script Creation Process

From Start to Finish

Ranges and Arrays

- Ranges are **sequence** of **numeric** values
  - **Single** range **1..10**
  - **Combined** ranges **1..20 + 40..50**
- Arrays are groups of values, can be single- or multi-dimensional
  - Implicit declaration - **$MyArray = 1, 2, 3, 4, 5**
  - Range declaration - **$MyArray = (1..5)**
  - Strongly typed - **[int[]] $MyArray = (1..5)**

Iterate Over Ranges and Arrays

- **ForEach-Object** or for short **ForEach**
- It can iterate over **ranges**, **arrays**, and **file contents**
- Two usage formats

```
...
10..20 | ForEach {"192.168.1.$_"}
...
$nodes = (10..20)
ForEach ($node in $nodes) {
"192.168.1.$node"
}
...
```

Goal and Self-Restrictions

- Start with a **clear idea** of what you would like to achieve
- Don't even start if
  - You will use the end-result **one time**
  - There is an **easier way**, do it, even on a regular basis
  - Always bear in mind the **ROI**
- Resist the temptation to **go big** since the very beginning
- Start small, with baby-steps. It is easier to debug if you like

Approach

- Break the end-task on **simple steps**
- Code **one step** at a time
- **Test** each **individual** step. Test **often**
- Write scripts with **the other guy** in mind
- Very often the other guy, **it is you** after a while

Good Practices

- **Comment** (document) what you are doing
- Put comment even after **each section end**
- Declare **variable types** explicitly
- **Structure** your code for **readability**. Make it pretty

Help Sections

- .SYNOPSIS
- .DESCRIPTION
- .PARAMETER <Name>
- .EXAMPLE
- .NOTES
- .LINK

```
<#

.SYNOPSIS
Simple PowerShell script skeleton with help template

.DESCRIPTION
This script is a very simple representation of how a
typical PowerShell routine with help should look like.

.EXAMPLE
./Script.ps1

.NOTES
Put some notes here.

.LINK
http://some.internet-domain.com

#>
# TODO: Place your code bellow
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_comment_based_help?view=powershell-5.1

Sourcing vs Execution

- Execute a script in its own environment

```
.\Script.ps1
...
C:\Scripts\Script.ps1
```

- Source a script to make it part of another environment
  - In a session

```
. .\Script.ps1
```

  - In another script

```
. "C:\Scripts\Script.ps1"
```

Practice: Script Creation Process

Live Demonstration in Class

Summary

- Not everything can be or should be done (easily) on command line
- Currently there are many scripting options
- For the last several years PowerShell is the way to go
- Scripts can include individual commands and logic
- Scripts are used to encapsulate and automate set of related tasks

Resources

- Official PowerShell Documentation
  https://docs.microsoft.com/en-us/powershell/
- PowerShell Community
  https://powershell.org/
- Official PowerShell Community Blog
  https://devblogs.microsoft.com/powershell-community/
- Hey, Scripting Guy! (retired)
  https://blogs.technet.microsoft.com/heyscriptingguy/

Software University

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

Software University

SCHWARZ

LIDL

Kaufland

SCHWARZ Produktion

pre zero

Coca-Cola HBC Bulgaria

POKERSTARS | CAREERS

Postbank

Решения за твоето утре

AMBITIONED

DXC TECHNOLOGY

SOFTWARE GROUP

Bosch.IO

INDEAVR

Serving the high achievers

DRAFT KINGS

PHAR VISION

SmartIT

createX

SUPER HOSTING .BG

Educational Partners

License

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg