Introduction to Scripting and 


---

# Test 1 Review Weeks 1-5 -- Key Topics

### WEEK 1 - INTRODUCTION TO POWERSHELL AND ISE

1 Introduction to PowerShell and ISE

   a. PowerShell 1.0 standalone product- had to be installed on XP , Vista for Server 2003

   b. PowerShell 2.0 bundled with Windows Management Framework (MWF)2.0

      i. Integrated into Windows 7 and Server 2008 R2

      ii. Uses .NET 3.5

   c. PowerShell 3.0 bundled with WMF 3.0

      i. Integrated into Windows 8 and Sever 2012

      ii. Uses .NET 4.0

2 Project and Release dates

   a. PowerShell project nicknamed “Monad”

      i. PowerShell 1.0 copyright date 2006

      ii. PowerShell 2.0 copyright date 2009

      iii. PowerShell 3.0 copyright date 2012

   b. To find PS version use copyright date in logo or $PSversionTable

3 Advantages of PowerShell over other Scripting Languages

   a. Built on top of .NET object model gives it the ability to work with objects (large blocks of code containing data and execution code with built-in intelligence)

   b. Powerful compound commands using Pipes

   c. Short learning curve because of consistent naming convention

4 Learning PowerShell requires understanding 3 things:

   a. CmdLets –built-in commands - verb-noun format

   b. Parameters – additional information modifies input\output of cmdlets-begin with “-“

c. Variables – containers to store values-begin with "$"

5 Commands and Syntax

a. Get-Help – general search get-help *<name>*

i. Get-help *-eventlog -finds verbs –action to be performed

ii. Get-Help New-* -finds nouns –object of the action

b. Get-Eventlog –requires parameter -logname

c. Get- Member –used to find methods and properties of objects

d. $Xmas | Get-Member – to view properties and methods

e. New-Timespan -requires parameter –end

i. $Xmas = New-Timespan –End 2014/12/24 (a start parameter is not needed- the default is today's date)

f. ConvertTo-HTML –converts text file to HTML format for web page

g. Out-File –used to create a file and output text to it.

h. Double quotes – interpolate strings (replaces variable names in the string) and Single quotes – literal string (does not replace variables in the string)

**Week 2 -- WORKING WITH POWERSHELL CONSOLE**

1. 4 Levels of Script Execution Built-in Security Policies

a. The prompt opens in the users profile directory. **PS <user profile path> >.**

b. Script execution is disabled by default.

c. Double clicking on a **ps1** file, opens the default editor, rather than executing the script.

d. Executable Files do not run automatically in the working directory.

i. Get-ExecutionPolicy – displays current policy

ii. Set-ExecutionPolicy – changes the policy – must be administrator

| Level | Meaning |
|---|---|
| Restricted | Will not run scripts or configuration files |
| AllSigned | All scripts and configuration files must be signed by a trusted publisher |
| RemoteSigned | All scripts and configuration files downloaded from the Internet must be signed by a trusted publisher. |
| Unrestricted | All scripts and configuration files will run. Scripts downloaded from the Internet will prompt for permission prior to running. |

3. 4 levels of scope when setting policy – scope refers to who runs the policy and where the policy is stored.

| Scope | Administered By and Stored in |
|---|---|
| MachinePolicy | Administered and Stored in Group Policy Object |
| UserPolicy | Administered and Stored in Group Policy Object |
| Process | Administered by PowerShell and stored in the $PSExecutionPolicyPreference Environment Variable |
| CurrentUser | Administered by System and stored in Registry Key Current User |
| LocalMachine | Administered by System and stored in Registry Key Local Machine |

$Profile – variable stores path to Microsoft.PowerShell_profile.ps1

- not enabled by default

-perserves console settings for customization.

## WEEK 3 - VARIABLES

1. Introduction to Scripting
   a. scripts are text files listing commands as they would be typed on the command line
   b. because of powerfull scripting languages and speed of processors, little difference today between scripting and programming.
   c. Comparison between Scripting and Programming Languages

| Summary Table Comparing Scripting and Programming Languages | |
|---|---|
| **Scripting** | **Programming** |
| Needs Parent program | Independent program |
| Simple and flexible language | Complex and strict language |
| Dynamic data types and variables | Static data types and variables |
| Interpreted Line by Line | Compiled as a whole |
| Reprocessed each time | Processed once |
| Best for repetitive tasks | Best for unique problems |
| Moderate\Fast run speed | Fast\Very Fast run speed |

2. Commands and Syntax – useful commands

    a. Get-History – lists al commands in session –fast way to build a script

        I. Get-History > commands.txt

    b. $home = c:\users\loginName

3. Types of Variables

a. Automatic - created and changed automatically by PowerShell to store system information such as $PSHOME which stores the path to the Windows PowerShell installation and $PWD which stores the PowerShell prompt path.

b. Preference – created by PowerShell to store user preferences which contain default values that the user can change.

    a. User-Created – stores user created and modified information

    a. Environment – stores information used by the operating system and applications, such as $OS which stores the current operating system, $TEMP which stores the path to the temp folder

        i. New Cmdlets – not commonly used in scripts –use alternate methods

| Cmdlets | Description |
|---|---|
| New-Variable | Creates a new variable and can set special attributes |
| Set-Variable | Modifies the value of an existing variable |
| Clear-Variables | Clears the contents of an existing variable and leaves the variable. |
| Remove-Variable | Deletes and existing variable and contents |

3. All variables and aliases, functions and registry stored on different PSDrives

a. Use Get-PSDrive to see drives

b. 2 ways to view variables

I. Cd variable: the type Get-Childitem

II. Get-Variable

4. Creating Variables

a. $Firstname = “YourName”

b. 2 ways to see the type of object created

I. $firstname | Get-member

i. All strings have a property length

II. $firstname.GetType()

c. By default variable names are case insensitive and PowerShell creates an “untyped” variable based on user input. To create a “strongly typed” variable use “[]” before the variable name i.e. [double] $num = 34.6

| Common Data Types | |
|---|---|
| **Type of** | **Description** |
| [int] | 32 bit signed integer |
| [long] | 64 bit signed integer |
| [double] | Double-precision 64-bit floating point number |
| [single] | Single-precision 32-bit floating point number |
| [bool] | Boolean true or false value |
| [string] | Fixed-length string of Unicode characters |
| [DateTime] | Date and Time object |

5. Naming Variables

a. Use long descriptive names avoid short cryptic names. Join long names combining upper and lower case words, e.g. $FirstName

6. Testing if variable exists – 2 methods

a. Test-Path variable:\FirstName – gives true or false if variable exists

b. Get-Variable FirstName –will bedisplayed if it exists

7. Environment Variables

a. to view all environment variables Get-ChildItem env:

b. important environment variables

I. $env:OS –name of os build number

II. $env:NUMBER_OF_PROCESSORS – number of CPU cores\processors

III. $env:ALLUSERSPROFILE – c:\ProgramData

IV. $env:COMPUTERNAME - name of computer

V. $env:PATH – stores the locations PS looks for executable files

i. When folder on the path – don't need to use".\"

ii. Modify path $env:Path = $env:Path + "; c:\scripts"

c. create environment variable -- $env:FirstName

d. remove environment variable 2 ways

I. $env:FirstName = $null

II. Remove-Variable FirstName

8. Special Variables

a. $PWD –stores prompt path

b. $Profile –stores user specific settings of the PowerShell console

c. $ErrorActonPreference = "SilentlyContinue" – determines how PS will handle errors during script execution.

**Week 4 Pipeline**

1. Pipeline old UNIX concept to create compound commands. The output of the command on the left becomes the input for the command on the right of the pipe "|" character
2. PowerShell follows 3 specific rules when "parameter binding" in a pipeline
    a. Does the parameter accept pipeline input? (Not all parameters of cmdlets do)
    b. Does the parameter accept object type, or is it able to convert it?
    c. The parameter cannot already have been used.
3. Parameter binding can be completed in 2 ways:
    a. "by value", which means that the piped object must be of the same type as the parameter object or be able to be converted to it
    b. "by PropertyName", which means the piped object must have the same name as the parameter of the input command.

c. Troubleshooting pipeline problems is done using the Trace-Command

e.g. **trace-command -name parameterbinding -expression {get-process taskmgr | ` stop-process} -pshost -filepath debugTaskmgr.txt**

4. PowerShell pipeline works in 2 modes: sequential and streaming

a. sequential mode means that the entire collection of the command on the left is passed to the command on the right and the entire collection is worked on at once. This can freeze or crash system if the collection is large.

-some commands like Sort-object – can only work in sequential mode

b. streaming mode is more memory efficient and faster execution time because as the collection is created by the command on the left, each item in the collection is processed one at a time across the pipeline

e.g**. Out-Host –paging** - sends one page of information across the pipe for processing therefore only need enough memory to process one page.

5. Modifying pipeline output

a.

| Common Parameters[1] | |
|---|---|
| **Parameter** | **Meaning** |
| -whatif | Tells the cmdlet not to execute; instead it will tell you what would happen if the cmdlet were to actually run. |
| -confirm | Tells the cmdlet to prompt prior to executing the command. . |
| -verbose . | Instructs the cmdlet to provide a higher level of detail than a cmdlet not using the verbose parameter. |
| -debug | - Instructs the cmdlet to provide debugging information. |
| -erroraction | - Instructs the cmdlet to perform a certain action when an error occurs. Allowable actions are: continue, stop, SilentlyContinue, and inquire. |
| -errorvariable | Instructs the cmdlet to use a specific variable to hold error information. This is in addition to the standard $error variable. |
| -outvariable | Instructs the cmdlet to use a specific variable to hold the output information. |
| -outbuffer | Instructs the cmdlet to hold a certain number of objects prior to |

[1] Wilson, Ed (2010-02-19). Windows PowerShell™ Scripting Guide (p. 12). Microsoft Press. Kindle Edition.

| | |
|---|---|
| | calling the next cmdlet in the pipeline |

b. Using the Tee-Object and Pass Through Parameters

1. since the pipeline works in a linear fashion, it is difficult to record log files at the end on the pipe when using commands like Stop-Process because there are no processes to record by the end of the pipeline

Eg. **Get-Process mspaint| Stop-Process | Out-File .\processes.txt – file is entry**

**Use Tee-Object to create a file in the middle of the pipeline line**

**Get-Process mspaint| Tee-Object -file kill.log |Stop-Process** - this creates log file of stopped processes before stopping them. Pipeline is working in 2 directions.

2. alternative approach to use –passthru parameter – stores a system information generated by the pipeline, in a separate area of memory, so it can output the info at the end of the pipeline

e.g. **Get-Process mspaint| Stop-Process –passthru > kill.log** - uses file redirection to save system info to a file

6. Measuring and Comparing Objects

a. Measure-Object cmdlet calculates three types of measurements on objects. It can count objects and calculate the minimum, maximum, sum, and average of the numeric values. For text objects, it can count and calculate the number of lines, words, and characters. (simple calculations)

e.g. **Get-ChildItem –recurse |Measure-Object –property length –min –max –average –sum**

**b.** Compare-Object cmdlet knows how to loop over the objects or collections, without having to write an iterative control structure. One object is declared a "reference set" which Compare-Object uses to identify the "difference set".

e.g. **Compare-Object -referenceobject $(get-content .\echoname.cmd)↵**

**–differenceobject $(get-content .\echoname.ps1) –includeequal**

-communicating with hardware – 2 methods WMI and CIM. –both provide the same functionality

CIM is better because platform independent, and better help files

Get-CIMclass Win32* -methodname rename

-to talk to hardware use Get-CIMinstance

Type: $drive = Get-CIMInstance Win32_LogicalDisk | Where-Object {$_.DriveID –eq 'C:'}

```
PS C:\Windows\system32> $drive

DeviceID      DriveType     ProviderName      VolumeName        Size          FreeSpace
--------      ---------     ------------      ----------        ----          ---------
C:            3                               Operating System 319965622272 16624648...
```

Figure 1: Using Where-Object to return just Hard Drive Info

Notice the volumeName property is empty. We can change the information by using the Set-CIMInstance cmdlet

Type: **$drive | Set-CIMInstance –arguments @{VolumeName = 'Operating System'}** –you must be logged in as administrator)

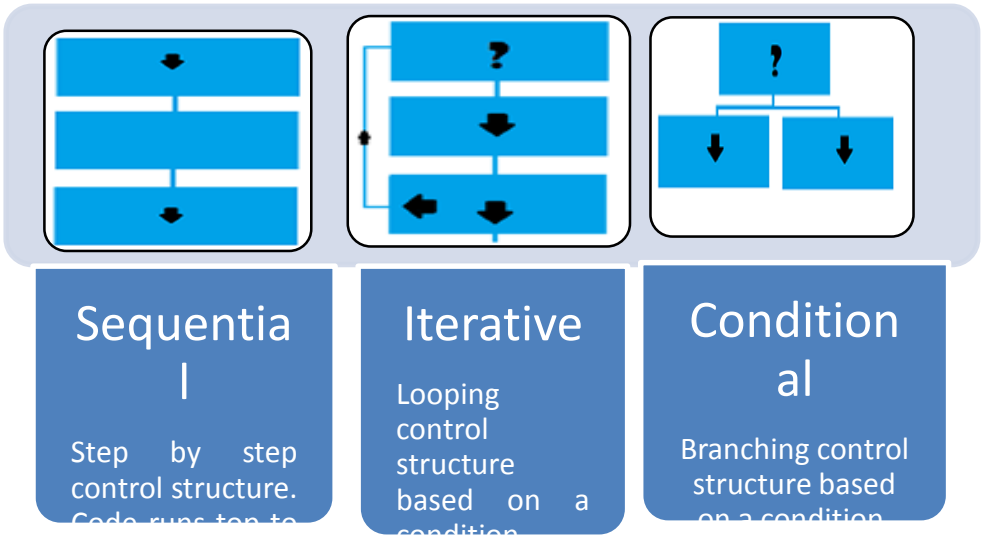
```
PS C:\Windows\system32> $drive

DeviceID      DriveType     ProviderName      VolumeName        Size          FreeSpace
--------      ---------     ------------      ----------        ----          ---------
C:            3                               Operating System 319965622272 16624648...
```

Figure 2: changing Drive Info with Set-CIMInstance

## Week 5 Control Structures

**1.** 3 types of control structure




es

a. All loops have conditions – tested each time through the loop, if condition is true the loop continues, when condition false loop stops and program flow follows along same path. – loop without exit condition is an "infinite loop"

b. All conditional structures have conditions, but only tested once and if false an alternate path is chosen

2. Goal of course is to script using "subprogramming" - different logic contained in different scripts and called by main program. When subprogram called PS give control to the program until the last line of script, then PS takes-back control.

3. Iterative (Looping or repetitive) control structures.

a. For($i=<value>; $i –le 5;$i++){Write-Host $i} –if condition true loop continues and carries out commands in script blocks. – ideal for repetitive operations – commonly used in creating\removing files\folders.

b. Foreach($<item> in $ <collection>){……..} – foreach cycles through a collection of objects using a built-in for loop to carry out commands in the script block on each item in the collection. – ideal for repetitive operations with collections - commonly used to cycle through a list of servers, performing the same operation on each.

c. Do {….} while(condition is true) - condition tested at bottom of loop – ideal for operations when you want code to run at least once, even if false - commonly used to monitor if an application is running.

d. While(condition is true) – condition tested at top of loop thus if condition is false the loop will not run – ideal for operations when you want code to run if true and you don't know how many times the loop needs to run.

e. Do {….} until(condition is true) – opposite of the do while –loop runs as long as condition is false and stops when the condition is true. – commonly used to give user a yes or no option to continue an operation

4. Conditional control structures allow scripts to made decisions and follow different code paths based on that decision. – good for error checking and building logic into scripts.

a. Conditional statements are based on comparison and logical operators

**Some Comparison Operators**

| Operator | Description |
|---|---|
| -eq | Equal to |

**Logical Operators**[2]

| Operator | Description |
|---|---|
| -not | Not |

[2] PowerShell Pro, "Conditional Logic" at http://www.powershellpro.com/powershell-tutorial-introduction/powershell-tutorial-conditional-logic

| | |
|---|---|
| -lt | Less than |
| -gt | Greater than |
| -ge | Greater than or Eqaul to |
| -le | Less than or equal to |
| -ne | Not equal to |

| | |
|---|---|
| ! | Not |
| -and | And |
| -or | Or |

5. Conditional statements

a. if(condition is true){……} e.g. if("$pwd" –ne "$DocumentsPath){Write-Host "Not in documents path"} - ideal for testing and continuing on same sequential path

b. if (condition is true){ …..}else{……} - ideal if you have two different paths to follow based on condition, if true the first script block is run, if false the else script block is run.

c. if elseif else – the elseif and else are optional command words.
if(condition is true){…….} – provides a code block based on a true condition
elseif(condition is true){…….} – provides a nested if alternative code block
else{condition is true} – provides a default true statement

The if elseif else command is ideal for nested if statements. Easier to read and less error prone. If the first condition is false, then the command flow goes to the elseif and tests if true, if it is, it runs the script block, if false, it continues on testing each condition until it reaches the else clause which is the default behavior and will always be true.

6. Switch

a. very compact command and is best option for large number of nested options or menus.

b. Switch syntax:

```
switch (expression)
{
 (test) {code block}
 value {code block}
 default {code block}
}
```

The expression is what the switch statement is using for testing. The value can be an expression or a text string that is being used for comparison. If the value tests to TRUE the code block is run. If not the flow goes to the next comparison value. The switch statement is handy if you need to find multiple TRUE statements. The last statement is a default code to run if all comparisons are FALSE.

# Study Procedure for Test 1

1. The notes provided above are not study notes, they are topical summaries of important material in weeks 1-6 for which you may be tested. Read the notes provided and if you are unsure of its meaning or command syntax, refer to the lecture notes and add the information to the notes section.

2. Study the lecture notes and make sure you understand the command syntax. If uncertain, add the command syntax to your reference sheet (1, 8 ½" X 11", handwritten sheet, two sided).

3. Use your text book as a reference guide, if understanding or syntax is unclear.

4. Practice writing simple control structures using the examples in the lecture notes and textbook. Coding is a large part of the test.

5. Try answering the questions in this review exercise. This review is to help you prepare for the test and get familiar with the types of questions you may be asked. If you are having problems completing this review exercise, email your instructor for help and guidance.

**Typical Test 1 Questions. – Total marks 32**

**8 Multiple Choice questions – 1 mark each --- 8 marks total**

1. PowerShell 4.0 version would have a copyright date of which of the following:

   a) 2012

   b) 2011

   c) 2012

   d) 2014

2. Understanding PowerShell requires understanding which of the following:

   a) Cmdlets

   b) Variables

   c) Parameters

   d) All of the above

3. PowerShell commands are based on which of the following:

   a) Verb-Noun syntax

   b) Noun-Verb syntax

c) Command-Noun syntax

d) Verb-Command syntax

4. The PowerShell command to read the contents of a file is which of the following:

   a) Read-Content

   b) Get-FileContent

   c) Get-Content

   d) None of the above

5. If you executed the following script on the command line using the syntax: **.\MyScript.ps1 "*.txt" "Documents" "Backup".** The argument "Backup" would have a value of $ARGS[3].

   a. True

   b. False

6. The code **for($i=0;$i -lt 5;$i++){Write-Host "Hello"; $i=$i+1}**will display the word "Hello" 3 times.

   a. True

   b. False

7. PowerShell 2.0 is integrated into the Vista operating system.

   a. True

   b. False

8. PowerShell pipeline sequential mode is more memory efficient.

   a. True

   b. False

**Practice Coding Questions**

9. Write short script which will do the following. Create a variable to hold the path **c:\users\public\documents\backup** . Create another variable to hold the path the file **$Home\documents\ Win213Notes.txt.** Test to see if the file exists, if it exists, move the file to the backup directory. If the file does not exist, display a message "File not found".

10. Write a short script which will do the following. Create 4 variables to hold each of the following names, Kirk, Spock, Bones and Scotty respectively. The user will be prompted, **"Who is the first officer on the Enterprise?"** Use an if elseif else statement to test the user response. If the user guesses the correct answer (Spock) the script should display **"Congratulations".** If the guess was incorrect the script should display a message **"Wrong: Kirk was the captain", "Wrong: Bones was the doctor"," Wrong: Scotty was the engineer"** respectively. If the user enters any other name than above, the script should display the message **"Unknown person ".**

11. Write a short menu that will display the following:

```
=======================================
1.  Find small files on drive
2.  Find largest file on drive
3.  Exit Program
=======================================
```

Use a Do loop and a switch statement to branch to each user selection. If the user does not make the correct selection, the script should display the message "Invalid selection. Try again". After making a selection the script should the menu. Assume all files are in the same directory, design your menu so that selection 1 calls the file **Get-SmallFiles.ps1** and selection 2 calls **Get-LargeFiles.ps1.**

**1 Find the Coding Errors Question --- 5 marks**

12. Find the errors in the following code and write the line number of the error in the left column and write the corrected line in the right column. There are no logic errors, only syntax and typos.

```powershell
###############################################################################################################
## FileName: Lab2_LearnName_MakeDir.ps1
## Author:  <YourName>   Date: <Today's date>   DateLastModified: <Today's date>
## Purpose:  Creates a script repository in the documents folder with weekly subfolders giving user the option
## Purpose:  to populate the subfolders with script files.
## Syntax: Lab2_LearnName_MakDir2.ps1  Source_FilePath Target_subdirectory
###############################################################################################################
###############################################################################################################
#>
Clear-Host

##Create variables for documents folder path, Win213, folder path and Scripts folder path
###############################################################################################################
$DocPath = "c:\users\dhr\documents"
$Win213Path = "c:\users\dhr\documents\Win213
$ScriptsPath = "c:\users\dhr\documents\Win213\Scripts"

##Create variables to hold the collection of folder names for Win213 folder and Scripts folder
###############################################################################################################
$Win213Folder = "WIP","Lec","Lab","Scripts"
$ScriptsFolder = "Wk1","Wk2","Wk3","Wk4","Wk5","Wk6","Wk7","Wk8","Wk9","Wk10"


##Test to see user is in the Documents folder and if not display a message if they want to quit, if NO changes to
## Documens folder and creates the Win213 directory
###############################################################################################################

$DirPath = Get-Location
if($DirPath -ne $DocsPath) {

        Write-Host "This script only runs in the documents directory. Change to the documents directory and continue? Press CTRL + C to exit or "
        start-sleep -s 5
        Set-Location $DocsPath
        $Win213 = New-Item -path . -ItemType Directory -Name "Win213"


}


## Cycle through the folder names in the Win213 folder to create subfolders using a foreach loop
###############################################################################################################
Foreach($dir in $Win213Folder){
    New-Item -path . -name "$dir" -ItemType Directory
}

##Cycle through the folder names to create the subfolders of the Scripts directory
###############################################################################################################
Foreach($dir in $ScriptsFolder) {
    New-item -path $ScriptsPath -name $ScriptsFolder -Itemtype Directory

}


##Do a recursive search of the documents folder to find all files PS1 and CMD and sort by lastwritetime and place in variable
###############################################################################################################

$Files = Get-ChildItem -path $docPath -recurse -include "*.ps1","*.cmd"
```

| Line of Error | Corrected Line |
|---|---|
| | |
| | |
| | |
| | |
| | |

**1 Short Answer Question – 5 marks**

13. Explain 3 features you like about PowerShell and its scripting environment.

-Make sure you provide sufficient detail to earn 5 marks. A short answer question is about 4-6 sentences long.

-other short answer topics are:

1. the differences between scripting and programming
2. -types of variables and how they are used
3. -types of control structures and when you would use each type
4. -what are cmdlets, variables and parameters and how are they used in scripting.

**During Test 1:**

1. You are allowed one reference sheet handwritten which will be turned in at the end of the test
2. You will be allowed to use PowerShell console, Get-Help, and your ISE during the test
3. You have the full class time to complete the test. When completed, make sure you click the SAVE and SUBMIT button at the end of the test. When you finish the test, please leave quietly.