Powershell

Functions, Parameters, User Input, Providers, WMI Objects
COMP2101 Fall 2017

Script Parameters

- Scripts can have parameters
- Use the param statement as the first line in your script to add parameters to your script
- The param statement adds variables to your script with the variable names being the parameter names
- The value of the parameter variables comes from the command line when the user enters those parameters
  param ($MyParameter, $AnotherParameter)
- Multiple parameters are separated by commas, and each parameter can have a type, default value, and additional attributes

Function Parameters

- Parameters can be specified for functions using the param block at the start of a function definition
  e.g.

```
function myfunc {
    param ([int]$myinteger = 1,
           [string[]]$mystrings,
           $something)
    ...
}
```

- Types and default values can be specified when defining parameters

Parameter Attributes

- Parameters can have attributes specified using the [parameter()] declaration immediately preceding the name of the parameter, this makes them into "Advanced Functions" and means you can use the common parameters with them
  e.g.

```
function myfunc {
    param ([parameter(Mandatory=$true,
                      Position=0,
                      ParameterSetName="MySet1",
                      ValueFromPipeline=$true,
                      ValueFromPipelineByPropertyName=$true,
                      HelpMessage="The MyParameter parameter can
have any value you like",
                      Alias=("mp","MyParameter"))]
           $MyParameter)
    ...
}
```

- Various validation attributes are available to further enforce parameter rules, see help about_functions_advanced

Parameter Attribute Table

- The attribute table, found in cmdlet help, is a detailed listing of the parameters a cmdlet will accept and how to specify them in a command

- help -full or -parameter or -online will show the parameter attribute table

- e.g

```
-path <string[]>
    Specifies a path of one or more locations. Wildcard characters are
    permitted. The default location is the current directory (.).

Required?                    false
Position?                    1
Default value                Current directory
Accept pipeline input?       true (ByValue, ByPropertyName)
Accept wildcard characters?  true
```

Common Parameters

- Cmdlets support common parameters which allow generic specification of common options

- Common parameters can be added to your scripts automatically by adding CmdletBinding() to your script or function before the param statement

- verbose (vb - $verbosepreference, used with write-verbose)
  debug (db - $debugpreference, used with write-debug)

- warningaction (wa - $warningactionpreference)
  erroraction (ea - $erroractionpreference)
  warningvariable (wv)
  errorvariable (ev)
  outvariable (ov)

- whatif (wi)
  confirm (cf)

- For more common parameters or more detail, refer to help about_commonparameters

Parameter Exercises

- Save the example below to a file
- Try running the command without any parameters, then with one of the parameters, then with both
- Try running the command without putting in the paramter names

```
Param ([Parameter(Mandatory=$true,position=1)][string]$SourceFile,
       [Parameter(Mandatory=$true,position=2)][string]$DestinationFile )
"SourceFile was '$sourcefile'"
$objtypename = $sourcefile.gettype().name
"SourceFile was a $objtypename object"
"DestinationFile was '$destinationfile'"
$objtypename = $destinationfile.gettype().name
"DestinationFile was a $objtypename object"
```

Working With Files Example

- This example will show a gridview listing of large document files in a folder specified by the user, having a minimum size specified by the user, with both parameters available to use from the command line
- It shows the script having 2 parameters, and a function with one parameter

FILE: bigdocs.ps1

```
param ([String]$Path=".", [long]$MinimumSize = 0)
function Get-Docs ([string]$Path=".") {
    Get-ChildItem -Path $Path `
                  -Include *.txt,*.doc,*.docx,*.pdf,*.xls,*.ppt,*.ps1 `
                  -Recurse `
                  -ErrorAction SilentlyContinue
}
Get-Docs -Path $path |
    Where-Object { $_.length -ge $minimumsize } |
    Select-Object FullName, LastAccessTime, Length |
    Sort-Object -Descending Length |
    Out-GridView
```

User Input

- User input can be obtained using the read-host cmdlet e.g.

```
$UserInput = read-host
[int]$Num = read-host -prompt "Give me a number "
$pass = read-host -prompt "Password: " -AsSecureString
```

- SecureString objects are designed to be used as part of credentials objects

  Extracting the original string from a securestring:

  ```
  (New-Object PSCredential -ArgumentList
  "someusername",securestringvariable).GetNetworkCredential().Password
  ```

- get-content will get data from a file as a generic object array

User Input Exercise

- Recreate the rolldice script from our bash lessons in powershell

Working With URLs Example

- Sometimes you want to retrieve files from web sites in a script
- This example retrieves a file from the course github repository and puts it into the current directory

FILE: gethubscript.ps1

```
param(
    [parameter(mandatory=$true)][string]$scriptname
)

$url = "https://github.com/zonzorp/COMP2101F15-01/raw/master/
powershell/$scriptname"
$localdir = pwd
$localfile = "$localdir/$scriptname"
$webclient = New-Object -TypeName system.net.webclient
$webclient.UseDefaultCredentials = $true
$webclient.DownloadFile($url, $localfile)
```

Providers

- Providers allow us to use drive name semantics to access different types of storage spaces
- get-psproviders shows the list of providers currently loaded in memory
- get-psdrive shows the list of drives using currently loaded providers with some summary information, very limited compared to WMI classes
- get-childitem (ls) can be used to view what is accessible via the providers by using the provider name as a drive name (e.g ls env: variable: alias: function:)
- Creating items of the types stored by these providers automatically stores them in that provider's storage

Storage Report (df)

- Compare the following two methods of viewing disk usage
- Note the additional information available when using WMI objects vs. using the output of cmdlets which already trim objects retrieved

```
get-psdrive
get-psdrive -psprovider filesystem
get-psdrive -psprovider filesystem |
    where-object {$_.used -or $_.free}


get-wmiobject -class win32_logical_disk
gwmi -class win32_logicaldisk |
  ? size |
  ft -auto deviceid, size, freespace, providername
gwmi -class win32_logicaldisk |
  ? size |
  select-object deviceid,
                @{n="Size(GB)";e={$_.size/1gb -as [int]}},
                @{n="Free(GB)";e={$_.freespace/1gb -as [int]}},
                @{n="% Free";e={100 * $_.freespace/$_.size -as [int]}},
                ProviderName |
  format-table -auto-size
```

Get-WMIObject

- Get-WMIObject retrieves many types of system information objects, gwmi is an alias for get-wmiobject

- gwmi -list shows a list of the retrievable objects, add a wildcarded word to the command to limit the output based on the class name
  e.g gwmi -list *adapter*

- WMIExplorer and the online resources from blackboard are also good places to discover useful WMI classes

Some Interesting WMI Classes

- win32_operatingsystem
  win32_bios
- win32_processor
  win32_cachememory
  win32_physicalmemory
- win32_logicaldisk
  win32_diskdrive
  win32_diskpartition
- win32_videocontroller
  win32_desktopmonitor
- win32_networkadapter
  win32_networkadapterconfiguration
- win32_printer
- win32_usbcontrollerdevice