PowerShell Files and Directories

SCRIPTING ESSENTIALS

DR. BURKMAN

Rethinking String slices

```
cls
$mystring = "Apple Banana Carrot"

$athing = $mystring[0..5]
Write-Host $athing
Write-Host $athing.GetType()
#This isn't a string

Write-host "-----------------"

$athing = $mystring.Substring(0,5)
Write-Host $athing
Write-Host $athing.GetType()
```

Rethinking String slices

```
cls
$cat = "123456789"

$catlen = $cat.Length
$catpos = $catlen - 5
$strlen = 5

$catpos = 2
write-host $cat.Substring($catpos,$strlen)
```

Arrays of arrays

To put an array inside another array you must put a comma in front of it.

```
cls
$a = "dog"          #$a is a string
$a = @($a)          #basically convert string to array
$b = @("cat")       #loading a string into an array

$j=@()              #init an array

$j += ,$a           #add array a to array j
$j += ,$b           #add array b to array j

write-host $j[0]

$j[0] += "mouse" #add a second element to array j

write-host $j[0]
write-host $j[0][1]
```

A Word on Splits

This is another area where PowerShell is inconsistent.

.split is not really the same as -split. Stick with .split for this class but be aware that PowerShell may return more elements than our previous programs.

One strategy for this is to check how many items are in the new array created by .split by using .count. Then manually iterate through the indexes (dog[0] then dog[1] etc) until you find the chunk you want.

You can reduce many of the unnecessary indexes by formatting your .split like this:

```
$split1 = $dog.Split("whatever",[System.StringSplitOptions]::RemoveEmptyEntries)
```

A Word on Splits

```
 $dog = "abc      d e"

 $split1 = $dog.Split("
",[System.StringSplitOptions]::RemoveEmptyEntries)

 foreach ($i in $split1)
 {
    Write-Host $i
 }
```

Concatenating Paths

Easy! Just use the + symbol. Pay attention to making sure that there is a back slash \ between each element when you join them

Unlike Python we can just use a single back slash

Working Directory

Find the working directory with Get-Location

Set the working directory with Set-Location

```
Get-Location
set-location "C:\Users\Burkman\Desktop\"
Get-Location
```

You can use the system variables to dynamically set the path on Windows:

```
$user_path = "C:\Users\" + $env:UserName + "\Desktop"
Set-Location $user_path
```

Checking for Directories

Use Test-Path to find out if an item exists. It has various switches you can use

- https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/test-path?view=powershell-5.1

```
Test-Path "C:\Users\Burkman\Desktop\"
Test-Path "C:\Users\Burkman\Desktop\b.py"
```

Use New-Item to make an ItemType. File and Directory are common data types, but there are more, and many switches you can use

- https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Management/New-Item?view=powershell-5.1

Making Directories

Use New-Item to make an ItemType. File and Directory are common data types, but there are more, and many switches you can use

- https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Management/New-Item?view=powershell-5.1

```
Test-Path "C:\Users\Burkman\Desktop\bunny\bunny.txt"
New-Item -Path "C:\Users\Burkman\Desktop\" -ItemType Directory -Name bunny
New-Item -Path "C:\Users\Burkman\Desktop\bunny" -ItemType File -Name bunny.txt
Test-Path "C:\Users\Burkman\Desktop\bunny\bunny.txt"
```

You can also just use mkdir to make a new directory. That's my favorite. In fact most of those BASH / command line commands will work.

Stopping Unwanted Output

Making things returns some execution report data. We can stop this by piping our work through Out-Null

```
Test-Path "C:\Users\Burkman\Desktop\bunny\bunny.txt"
New-Item -Path "C:\Users\Burkman\Desktop\" -ItemType Directory -Name bunny | Out-Null
New-Item -Path "C:\Users\Burkman\Desktop\bunny" -ItemType File -Name bunny.txt | Out-Null
Test-Path "C:\Users\Burkman\Desktop\bunny\bunny.txt"
```

Reading a File Using Get-Content

```
cls
#make path to the user desktop
$user_path = "C:\Users\" + $env:UserName + "\Desktop"
Set-Location $user_path

$file = ".\vkissoff.txt"

foreach ($line in Get-Content $file)
{
    Write-Host $line
}
```

Tracking Line Numbers

You can use .readcount and IF to skip one or more rows

```
#make path to the user desktop
$user_path = "C:\Users\" + $env:UserName +
"\Desktop"
Set-Location $user_path

$file = ".\vkissoff.txt"

foreach ($line in Get-Content $file)
{
    if ($line.readcount -eq 1)
    {
        continue
    }
    Write-Host $line
}
```

Writing a File with Out-File

You have to use –encoding asci for this to properly create csv files. Use -append to append to an existing file or leave if off to make a new file. Use Write-Output, not Write-Host

```
 cls
write-output ("Name,Address,City,State") | Out-File jim.csv -encoding ascii

$name = "Jim"
$address = "123 Oak St."
$city = "Stillwater"
$state = "OK"

Write-Output ($name + "," +
$address + "," +
$city + "," +
$state) | Out-File jim.csv -encoding ascii -Append
```

	A	B	C	D
1	Name	Address	City	State
2	Jim	123 Oak St.	Stillwater	OK

Compressing a File (or Files)

```
cls
$user_path = "C:\Users\" + $env:UserName + "\Desktop"
Set-Location $user_path

$my_zip = @{
    Path = ".\jim.txt", ".\vkissoff.txt" #separate files with a commma
    CompressionLevel = "Fastest"
    DestinationPath = "myzip" #don't add .zip to the file name
}
Compress-Archive @my_zip
#Compress-Archive @my_zip -Force #If you need to overwrite the zip file
#Compress-Archive @my_zip -Update #If you need to add/update a zip file
```

Expanding a Zip File

```
Expand-Archive -Path '.\myzip.zip' #-DestinationPath .\dog
#DestinationPath will make a directory if specified
#Only the zip contents will go in the new directory
#You can add –Force to this as well
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.archive/expand-archive?view=powershell-7.1