

# Useful PowerShell Commands

Placeholder

## Select-String is the Grep equivalent

Examples:

```
# Searching for multiple patterns at the same time
Select-String -Path "*.txt" -Pattern "Pattern1","Pattern2","Pattern3"

# Only return the first 10 results
Select-String -Path "*.txt" -Pattern "Pattern1","Pattern2","Pattern3" | Select-Object -First 10

# Searching for IP addresses
Select-String -Path "*.log" -Pattern '\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b' | Select-Object -First 10
```

## Uptime

The script below will give you the uptime in any version of PowerShell.

```
Get-CimInstance -ClassName Win32_OperatingSystem | Select LastBootUpTime
```

The [Get-Uptime](#) cmdlet was introduced in PowerShell 6.0.

```
Get-Uptime
```

## Format processes by start date

This command will show a lot of errors if you're not running PowerShell as Administrator.

# Active Directory Account Information

This command will show you the date of the last password set for a user.

```
Get-ADUser -Identity [USERNAME] -properties * | select accountexpirationdate, accountexpires,  
accountlockouttime, badlogoncount, padpwdcount, lastbadpasswordattempt, lastlogondate, lockedout,  
passwordexpired, passwordlastset, pwdlastset | format-list
```

Sources:

[PowerShell Format-Table](#)

## Active Directory Account Password Expiration

The old way:

```
net use userName /domain
```

The PowerShell way:

```
Get-ADUser -identity userName -Properties "DisplayName", "msDS-UserPasswordExpiryTimeComputed" |  
Select-Object -Property  
"Displayname", @{Name="ExpiryDate"; Expression={[datetime]::FromFileTime($_."msDS-  
UserPasswordExpiryTimeComputed")}}
```

## Active Directory OU Account Password Expiration

```
Get-ADUser -filter * -SearchBase "OU=Management,OU=ADPRO Users,DC=ad,DC=activedirectorypro,DC=com"  
-Properties "DisplayName", "msDS-UserPasswordExpiryTimeComputed" | Select-Object -Property
```

```
"Displayname",@{Name="ExpiryDate";Expression={[datetime]::FromFileTime($_.msDS-UserPasswordExpiryTimeComputed)}}}
```

## View physical network interfaces

```
# Show all physical devices
```

```
Get-NetAdapter -Physical | Sort-Object -Property MediaType,Name | Format-Table  
ifIndex,MediaType,InterfaceMetric,Name,InterfaceDescription,Status,MacAddress,LinkSpeed
```

## Get interface metrics

```
# IPv4 - Display interfaces sorted by metric and alias
```

```
Get-NetIPInterface -AddressFamily IPv4 | Sort InterfaceMetric,InterfaceAlias
```

```
# IPv6 - Display interfaces sorted by metric and alias
```

```
Get-NetIPInterface -AddressFamily IPv6 | Sort InterfaceMetric,InterfaceAlias
```

```
# All - Display interfaces sorted by metric and alias
```

```
Get-NetIPInterface | Sort InterfaceMetric,InterfaceAlias
```

## Set interface metrics

The following commands will set Ethernet interfaces to be preferred over wireless interfaces by manipulating the InterfaceMetric of each device. If there are more than one Ethernet and/or Wireless interface on the machine, you may want to adjust these metrics further to provide a more detailed use order.

### **Ethernet first, then wireless:**

```
# Set Ethernet devices interface metric to 11
```

```
Get-NetAdapter -Physical | Where {$_.MediaType -eq "802.3"} | Set-NetIPInterface -InterfaceMetric 11
```

```
# Set Wireless devices interface metric to 12
```

```
Get-NetAdapter -Physical | Where {$_.MediaType -eq "Native 802.11"} | Set-NetIPInterface -InterfaceMetric 12
```

### Wireless first, then Ethernet:

```
# Set Wireless devices interface metric to 12
```

```
Get-NetAdapter -Physical | Where {$_.MediaType -eq "Native 802.11"} | Set-NetIPInterface -InterfaceMetric 12
```

```
# Set Ethernet devices interface metric to 13
```

```
Get-NetAdapter -Physical | Where {$_.MediaType -eq "802.3"} | Set-NetIPInterface -InterfaceMetric 13
```

## Is your Office installation 32 or 64 bit?

```
# .platform value will be either x86 for 32-bit or x64 for 64-bit
```

```
$officeCheck = (Get-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\Office\ClickToRun\Configuration").platform
```

```
if ($officeCheck -eq 'x64'){
```

```
    Write-Output "Office is 64 bit."
```

```
}
```

```
else {
```

```
    Write-Output "Office is 32 bit."
```

```
}
```

## Exporting Event Logs using Out-HTMLView

You can use the Out-HTMLView module to view or save and view later.

```
$executionPolicy = Get-ExecutionPolicy
```

```
#Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force
```

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Unrestricted -Force
```

```
try { Import-Module -Name PSWriteHTML }
```

```
catch {
```

```
    Install-Module -Name PSWriteHTML
```

```
    Import-Module -Name PSWriteHTML
```

```

}

$lastHours = -4
$timeStamp = (Get-Date).ToString('yyyyMMdd_HH:mm:ss')
$systemEventLogFile = ("$(env:TEMP)\$(timeStamp)_eventlogs_system_Out-HTMLView.html")
$applicationEventLogFile = ("$(env:TEMP)\$(timeStamp)_eventlogs_system_Out-HTMLView.html")

Get-EventLog -LogName System -After (Get-Date).AddHours($lastHours) | Out-HTMLView -FilePath
$systemEventLogFile
Get-EventLog -LogName Application -After (Get-Date).AddHours($lastHours) | Out-HTMLView -FilePath
$applicationEventLogFile

Write-Host ("Event Logs for the last $($lastHours) hours saved to the following files:")
Write-Host ("$(systemEventLogFile)")
Write-Host ("$(applicationEventLogFile)")

#end

```

## List installed Windows Features

```
Get-WindowsFeature | Where-Object {$_.installstate -eq "installed"}
```

## CPU utilization

### Source

```

Get-Counter -ComputerName localhost '\Process(*)\% Processor Time' `
| Select-Object -ExpandProperty countersamples `
| Select-Object -Property instancename, cookedvalue `
| Sort-Object -Property cookedvalue -Descending | Select-Object -First 20 `
| ft InstanceName,@{L='CPU';E={$_.Cookedvalue/100}.toString('P')}} -AutoSize

```

Revision #20

Created 1 January 2021 23:38:57 by bluecrow76

Updated 16 July 2024 16:28:00 by bluecrow76