PowerShell Fundamentals

Labbmanual

september 2020
Fredrik Wall
fredrik.wall@retune.se

Innehåll

Introduktion

Hej,

välkommen till PowerShell fundamentals labbmanual.

Tanken med denna labbmanual är att den följer avsnitten som läraren pratar om och demar.

Läraren kommer att tala om vilka övningar och uppgifter ni ska göra när det är dags för att labba.

Stöter du på några frågor eller saker som verkar konstiga så fråga!

Tanken med labbarna är att man ska få en förståelse för hur saker fungerar, en förståelse för hur man tar reda på hur man löser saker och bygga vidare på vad läraren pratar och visar.

Labbmiljön

Labbmiljön som vi använder består av 3 virtuella maskiner.

10961C-LON-DC1 (lon-dc1)	- DC – Windows Server 2016
10961C-LON-CL1 (lon-cl1)	- Klient – Windows 10
10961C-LON-SVR1 (lon-svr1)	- Server – Windows Server 2016

Länken till din labbmiljö ska du ha fått innan annars får du den nu.

De flesta övningarna kommer att utföras från den virtuella klienten (lon-cl1).

Användarnamn: Administrator
Lösenord: Pa55w.rd

Behöver man logga in på datorn så är det följande som gäller:
Användarnamn: Administrator
Lösenord: Pa$$w0rd

Filerna som används i labbarna finns att hämta här:
https://github.com/fredrikwall -> PowerShell -> Fundamentals

Filer.zip

Skapa en katalog under c:\ som heter temp.
Spara filen under **c:\temp** så länge!

Materialet får inte olovligen kopieras.

Övning 1 - PowerShell konsolen

Tid: ca 10 minuter

I den här övningen bekantar vi oss med PowerShell konsolen.
Vi lär oss hur vi kan starta den på lite olika sätt, ändra färger och bra snabbkommandon.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Skapa genväg

Vi börjar med att skapa 1 genväg till PowerShell på skrivbordet.

1. Lättast gör vi detta genom att högerklicka på skrivbordet.
   Välj **New -> Shortcut**.

2. Och skriver ”**powershell.exe**”.

3. Döp den till **PowerShell ”Vad du vill”**, eller **”Vad du vill” PowerShell**.
   Till exempel “**PowerShell” eller “Powered by PowerShell”**.

4. Dubbelklicka på genvägen för att testa att genvägen fungerar.

   Nu har vi skapat en genväg till en standard PowerShell konsol.

   Stäng ner konsolen genom att antingen skriva exit eller klicka på krysset uppe i högra hörnet.

Uppgift 2 – Ändra startkatalog

Ändra i vilken katalog PowerShell startar

Det kan vara ganska smidigt att låta PowerShell starta i den katalog där vi har tänkt ha våra PowerShell script filer eller en jobbkatalog.

Till exempel **C:\scripts**, eller så vill man ha det på sin hemkatalog på en server.

1. Skapa en katalog direkt under **C:** som heter **scripts**.

   Detta gör du lättast genom att skriva:

   New-Item -Path C:\ -Name Scripts -ItemType Directory

2. Ändra så att genvägen startar i **c:\scripts**.

3. Starta PowerShell från genvägen.

Uppgift 3 – Ändra färger

Alla är vi vana vid olika färger i olika sammanhang.
Detta går att styra i PowerShell konsolen också.

Ändra till en textfärg som passar dig.

Detta gör du genom att klicka på PowerShell ikonen längst upp till vänster på konsolen. Välj properties och colors.

1. Skriv:

   Get-ChildItem -Path C:\Windows

   i konsolen och tryck **Enter**.

2. Känns den nya färgen bättre eller sämre än den tidigare?

Ändra till du känner dig nöjd.

Uppgift 4 – Använda snabbkommandona

Det finns en del bra snabbkommandon i PowerShell.

1. Öppna upp PowerShell konsolen genom att klicka på den första genvägen vi skapade.

2. Skriv:

   Start-Transcript -Path C:\Scripts\Minloggning.txt

3. Skriv:

   Get-Command

   Och tryck **Enter**.

4. Skriv:

   Get-Help

   Och tryck **Enter**.

5. Skriv **Get-ChildItem c:\windows** och tryck **Enter**.

6. Skriv **Clear-Ho**, tryck **TAB** och tryck **Enter**.

7. Tryck på **pil upp tangenten** 3 gånger.
   Tryck på mellanslag och skriv **Get-Alias** och tryck **Enter**.

8. Tryck på **pil upp tangenten** 1 gång och tryck **Enter**.

9. Avsluta PowerShell konsolen.

Slut på Övning 1!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 2 – Cmdlets

 **Tid:** ca xx minuter

I den här övningen börjar vi titta på Cmdlets.
En del av PowerShells inbyggda kommandon.

Vi lär oss bland annat hur man hittar hjälpen.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Hitta Cmdlets

1. Öppna upp PowerShell konsolen genom genvägen vi skapade i början av **Övning 1**.

2. Nu när vi har bekantat oss med PowerShell konsolen i **Övning 1** så kommer vi nu att bekanta oss med cmdlets i Övning 2.

3. Skriv **Get-Com** och tryck **TAB**, tryck sedan **Enter**.

4. Nu visas alla kommandon som vi kommer åt vi PowerShell.

5. Skriv **Get-Command -CommandType cmdlet**

6. Nu får vi upp alla cmdlets som finns tillgängliga på datorn.

7. Vi vill ta fram en lista på de cmdlets som tar fram saker.

8. Skriv **Get-Command -verb Get**.

9. Om vi i stället vill ta reda på vilka cmdlets vi har som rör services så får vi skriva **Get-Command -Noun Service**

10. Skriv **Get-Service**

Uppgift 2 – Hitta hjälp

1. Börja med att skriva **cls** eller **Clear-Host** och trycka Enter.

2. Skriv **Get-Help Get-Date -Detailed**

3. Om nu går upp en bit i all information som visas på skärmen så att du kommer upp till **Example 3** så hittar du bra information om hur man kan använda cmdleten **Get-Command**.

4. Samma gäller alla **Examples** som visas.
   Det ger en bra överblick över hur man kan använda cmdleten man letar hjälp om.

Uppgift 3 – Lägg till parametrar till en cmdlet

1. Skriv **Get-ChildItem C:\windows** och tryck Enter.

2. Nu lade vi till en parameter till cmdleten **Get-ChildItem**.
   En **positionsparameter**.

3. Om vi nu istället skriver ”**Get-ChildItem -Path c:\windows**” och trycker Enter.
   Så har vi gjort precis samma sak, men istället använt sökvägen som en **namngiven parameter**.

4. Just nu visar vi bara filer och kataloger i katalogen **c:\windows**.

5. Skriv **Get-ChiltItem –Path c:\windows –Recurse**.
   Nu visas alla filer i **c:\windows** och i alla underkataloger.

6. Och detta genom att vi använde **växeln recurse**.

7. Den kommer att hålla på läääänge, så tryck **Ctrl – C** för att avbryta.

8. Som ni säkert såg så var det något rött som flashade förbi.

9. Detta beror helt enkelt på att det är en katalog där vi som vanliga användare inte kommer in i.

   Så det är inget märkvärdigt med det.

10. Men för att slippa den så skriver vi:
    Get-ChildItem –Path c:\windows –recurse –ErrorAction SilentlyContinue

11. Och nu har vi använt en **allmän (common) parameter** också.

12. Nu tänker du att, det där med Access Denied till en folder bara är skitsnack och en bortdribbling.

13. Det kan man enkelt se genom att använda samma parameter, men istället skriva följande.
    Get-ChildItem –path c:\windows –recurse –ErrorAction Stop

14. Det vi gjorde är att ge parametern **–ErrorAction**, alltså vad göra vid Error, först värdet **SilentlyContinue** (låtsas som ingenting) som är ganska vanligt vid skriptning och i andra exemplet när vi synade mig fick parametern värdet **Stop** (stanna allt).

För att hitta parametrar till en cmdlet så skriver man:
Get-Help Get-ChildItem -Detailed

Sedan får man skrolla uppåt till **PARAMETERS**.
Under parameters hittar man alla parametrar som hör till en cmdlet
+ common parameters.

Uppgift 4 – Skriv ut information

En annan vanlig cmdlet är **Write-Output**.
Den används ofta till att skriva ut information om vad som händer till den som kör ett skript.

1. Skriv **Write-Output ”Hello World”** och tryck Enter.

Uppgift 5 - Testa om en sökväg finns

Det finns en massa färdiga kommandon för att underlätta för oss.
En av de mer användbara cmdletsen är Test-Path.

Vad tror du att den gör?
Använd hjälpen för att hitta informationen.
Skriv **Get-Help Test-Path**
Testa dig fram med full och detailed.

Skriv **Test-Path -Path C:\scripts**

1. Vad får du för svar?

2. Om du istället skriver **Test-Path -Path C:\ScriptS**

3. Får du samma svar?

4. Är **Test-Path** känslig för för stora och små bokstäver?

Slut på Övning 2!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrafrågorna nedan.

Extrauppgifter

I uppgift 5 använde vi cmdleten **Test-Path** för att se om en fil finns.
Det går att använda Test-Path mot annat än filsystemet och med andra rättigheter.

Frågor:

1. Om vi istället skulle vilja testa om vi hittar något i registret.
   Hur gör vi då?

2. Om du skulle vilja testa om en sökväg finns på en sökväg där din ”vanliga” användare inte har rättigheter.

   Vilken parameter kan du då använda?

Övning 3 – Aliases

Tid: ca xx minuter

I den här övningen tittar vi på PowerShells Aliases.
Andra namn, kortare namn för existerande kommandon.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Hitta Aliases

1. Öppna upp PowerShell konsolen genom genvägen vi skapade i början av Övning 1.

2. Skriv ”**Get-Alias dir**”

3. Nu får vi upp vilken cmdlet som Dir är ett alias på.

4. Skriv ”**Get-Alias ls**”

5. Nu får vi upp vilken cmdlet som ls är ett alias på.

6. Och som vi ser är det samma cmdlet i båda fallen, **Get-ChildItem**.

7. Om vi vill lista alla aliases som finns i PowerShell sessionen vi är i så skriver vi **Get-ChildItem Alias:** och trycker Enter.

Uppgift 2 – Skapa egna Aliases

1. Tänk om vi kunde få upp våra favoritprogram direkt ifrån PowerShell konsolen genom att bara skriva ett ord.

2. För oss administratörer som ofta tar skärmbilder för dokumentation och liknande så är Skärmklippsverktyget i Windows 10 en riktig skatt.

3. För att skapa ett alias på detta så skriver vi bara ”**Set-Alias Sax SnippingTool.exe**” och trycker Enter.

4. Skriv sedan ”**Sax**” och tryck Enter.

5. Nu startas Skärmklippverktyget.

6. Gör nu detta även för Notepad och döp aliases till t ex edit eller anteckningar.

7. Nu har vi skapat två aliases för två program.
   Men om vi istället vill skapa ett alias för en cmdlet så gör vi på samma sätt.

8. Skriv ”**Set-Alias Listafiler Get-ChildItem**”.

9. Nu har vi skapat oss en egen alias för **Get-ChildItem**.

10. Skriv ”**Listafiler -Path C:\windows**”.

11. Nu tänker du ”Haha, nu kom jag på en smart sak, jag skapar ett alias för att lista filerna i c:\windows, listawindows”.

12. Om vi då skriver:
    Set-Alias ListaWindows Get-ChildItem c:\windows
    och trycker Enter.

13. Nu får vi ett felmeddelande.

14. Detta beror på att man kan bara sätta aliases på cmdlets utan att sätta parametrar och funktioner.

15. Så vill man ha ett alias enligt ovan så får man skapa det på en funktion.

16. Stäng ner PowerShell konsolen.

Uppgift 3 – Spara Aliases

1. Öppna upp PowerShell konsolen.

2. Skriv nu **Sax**

3. Nu får vi ett felmeddelande som säger att PowerShell inte känner igen termen ’**Sax**’.

4. Detta beror helt enkelt på att alla aliases som man sätter själv endast är sparade i den PowerShell session som man jobbar i. Så avslutar man PowerShell så försvinner den.

5. Detta kan man lösa på lite olika sätt.

6. Skriv **Set-Alias Sax SnippingTool.exe** och tryck Enter.

7. Om vi nu exporterar alla aliases genom att skriva:
   Export-Alias minaaliases.

8. Så kommer vi att ha en fil som heter **minaliases** med **alla** aliases.
   Dock kan den filen ha sparats någon annanstans om vi inte stod i katalogen **C:\scripts** när vi sparade.

   En bra regel är att använda namngivna parametrar när man använder cmdlets och särskilt när man ska skapa eller spara saker.

   Export-Alias C:\Scripts MinaAliases

   Skulle vara en snyggare lösning.

9. För att slippa starta om PowerShell igen för att testa detta så skriver vi
   Remove-Item Alias:sax

10. Skriv nu **Sax** för att se att aliaset verkligen inte är där.

11. För att få tillbaka de aliases vi exporterade i punkt 7 så skriv:
    Import-Alias C:\Scripts\MinaAliases -Force

12. Skriv nu **sax** för att se att aliaset är tillbaka.

Slut på Övning 3!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrafrågorna nedan.

Extrauppgifter

Frågor:

1. Hur gör vi för att lista alla Alias till **Get-ChildItem**?
2. Hur vet vi att **Get-ChildItem** är en cmdlet och inte t ex en inbyggd funktion?
3. Är det bra att ha massa egna aliases när man ska börja dela med sig av skript till andra?
4. Varför skulle det kunna vara dåligt?

Övning 4 – Inbyggda funktioner

Tid: ca 5 minuter

I den här övningen går vi igenom hur man visar alla inbyggda funktioner.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Hitta inbyggda funktioner

PowerShell har många färdiga cmdlets och funktioner.
För den som använder dem är de i princip likadana.

Det som skiljer cmdlets från funktioner är att cmdlets är skriva i C# och funktioner består av PowerShell kommandon och PowerShell uttryck.

1. Skriv **Get-Command -CommandType function**

   Dock visar detta i vissa fall även cmdlets.

2. Men skriver man **Get-ChildItem function:**
   Så visas bara de funktioner som PowerShell är inlagda i functions driven.

Slut på Övning 4!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 5 – Variabler

Tid: ca 10 minuter

I den här övningen tittar vi närmare på variabler.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 - Lista innehållet i Windows katalog med en egen variabel

1. Skriv **$WindowsDir = ”C:\windows”**
2. Skriv **Get-ChildItem $WindowsDir**

Uppgift 2 – Använda variabel för servernamn

1. Skriv **$ServerName = ”lon-dc1”** och tryck Enter.
2. Skriv **Get-Process -computerName $ServerName** och tryck Enter.
3. Skriv **Get-Hotfix -ComputerName $ServerName** och tryck Enter
4. Skriv **Get-Service -ComputerName $Ser** och tryck TAB och tryck sedan Enter.

   Tanken är att du ska slippa skriva ut variabeln ServerName utan TABa dig till den.

Uppgift 3 - Hitta variabler

1. Skriv **Get-ChildItem Variable:** eller **Get-Variable**
2. Leta fram alla variabler som har med Server att göra.
3. Skriv **Get-ChildItem Variable:server*** eller **Get-Variable -Name Server***

Uppgift 4 – Ta bort variabler

1. För att hålla fint efter oss så ska vi ta bort våra två variabler som vi har skapat.

2. Skriv ”**Remove-Item Variable:WindowsDir**” eller
   Remove-Variable -Name “WindowsDir”

3. Skriv **Remove-Item Variable:ServerName** eller
   Remove-Variable -Name “ServerName”

Uppgift 5 – PowerShell version

En vanlig fråga när man håller på med PowerShell är vad är det för version.
Detta finns lagrat i en fördefinierad variabel som ett objekt.

1. Skriv bara **$PSVersionTable** och tryck Enter.

2. Vill man bara få fram vilken version det är så kan man skriva
   $PSVersionTable.PSVersion

3. Du kan även använda cmdleten **Get-Host** för att få fram PowerShell versionen.

Slut på Övning 5!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrafrågorna nedan.

Extrauppgifter

Skriv **Get-Item variable:* | Remove-Item -Force**

Fråga:

1. Vad hände?
2. Hur gör man för att det ska bli okej?

Övning 6 – Miljövariabler

Tid: ca 5 minuter

I den här övningen tittar vi på miljövariabler.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 - Använda miljövariabler för att ta fram information om kataloger

1. Tidigare har vi använt egen variabel för **C:\windows**. Det behöver vi egentligen inte göra då Windows har det lagrat i environment variabeln **windir**.

2. Skriv **Get-ChildItem $env:windir** och tryck Enter.

3. Skriv **Get-ChildItem $env:HOME** och tryck TAB och TABa 2 gånger och tryck Enter.

Uppgift 2 – Lista alla miljövariabler

1. Som med alias och function så ligger environment variabler som en psdrive. Detta innebär att vi kan skriva **Get-ChildItem env:**

Uppgift 3 – Ta reda på information om datorn

Nu vill vi använda miljövariabler för att få reda på saker om datorn.

1. Vi vill ha reda på följande saker:

   Datorns namn
   Antal processorer
   OS
   System Drive

2. Skriv ”**$env:**” och tryck TAB tills du fått tag på det du vill få information om och tryck sedan Enter och upprepa tills du har all information.

3. För att skriva ut detta på skärmen i ett skript använder man oftast cmdleten **Write-Output** och skriver då t ex
   Write-Output ”Datornamn: $env:COMPUTERNAME”

Slut på Övning 6!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 7 – Objekt

 **Tid:** ca 10 minuter

I den här övningen börjar vi titta på objekt.

Får du problem med något så fråga så går vi igenom det.

PowerShell är alltså objektbaserat.
Men det är lite svårt att ta på i början.

I denna övning ska vi förstå lite mer vad ett objekt är och varför PowerShell med objekt är bra om man jämför med t ex VBScript där output är endast text.

Uppgift 1 – Hur ser man vad saker är?

PowerShell tar hand om det mesta själv när det kommer till objekt och vilken typ av objekt det är.

För att ta reda på vilken typ ett objekt är så:

1. Skriv:
   $MyVariable = ”Hello World” och tryck Enter.

   Skriv sedan:
   $MyVariable.GetType()

   Då får vi fram att vår variabel är av typen String.

   Det vi gjorde var att vi valde att visa vårt objekt med metoden GetType istället för att bara visa objektet.

Uppgift 2 – Hur får man fram vilken information man kan få fram från ett objekt?

Hur skulle man då kunna veta att man kan anropa vår variabel som innehöll ett strängobjekt för att få ut vilken typ det var?

1. Skriv **$MyVariable | Get-Member**

   Nu listar vi medlemmar, egenskaper och metoder på vårt objekt som vi skapat i variabeln **$MyVariable**.

Uppgift 3 – Är alla egenskaper på ett objekt av samma typ?

När man t ex har tagit fram information om en fil så har man ett objekt och typen är FileInfo.

1. Skriv **Get-Item -Path C:\windows** och TABa dig fram till en dll fil. Första bästa går bra.

   Nu visas det som Microsoft tycker är viktigast.

2. För att se typen på hela objektet så skriver man enklast (före och) efter allt och lägger till .Gettype() efter).

   T ex **(Get-Item -Path c:\windows\twain_32.dll).GetType()**

3. Och för att se vilka egenskaper den har så gör vi som vi gjort på objekt tidigare.

   T ex **(Get-Item -Path C:\windows\twain_32.dll) | Get-Member**

   Kan man redan här se vilken typ en egenskap har?

Slut på Övning 7!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 8 – Arrayer

Tid: ca 10 minuter

I den här övningen tittar vi på Arrayer.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Hitta en Array

1. Skriv **$MyTest = ”Hej på dig”**

4. För att fråga om något är något så använder vi villkoret (condition) –is.

5. Skriv **$MyTest -is [array]**
   Är **$myTest** en array?.

6. Svaret blir False då **$MyTest** endast innehåller en sträng.

4. Skriv **$MyTest = ”Hej”,”på”,”dig”**

5. Skriv **$MyTest -is [array]**
   Är **$MyTest** en array nu?.

4. Denna gång har vi skickat in fler värden i variabeln **$MyTest** och därför är det numera en array.

5. Vill vi veta vad för typ en variabel är så kan vi skriva:

   $MyTest.GetType()

Uppgift 2 – Plocka fram första och sista underkatalogen i en katalog

1. Din arbetskamrat **Kjelle** kommer till dig och vill veta första och sista katalogen i profilkatalogen för användare.

2. Skriv **$MyProfileFolder = Get-ChildItem -Path $env:USERPROFILE**
   och tryck Enter.

3. Skriv **$MyProfileFolder[0,-1]** och tryck Enter.

4. Nu visar sig Videos och Contacts.

5. **Kjelle** blir eld och lågor och undrar om man inte kan se t ex när någon senast lade något i katalogerna.

6. Skriv **$MyProfileFolder[0,-1] | Format-List *** och tryck Enter.

7. Nu får ni fram massa information!

8. Om du skriver **$MyProfileFolder** så får ni fram alla kataloger och det ser ut som en ”vanlig” **Get-ChildItem** eller **Dir**.

9. Skriv **$MyProfileFolder | Format-List *** så får ni fram extra information om alla kataloger.

Uppgift 3 – Spara datornamn i en Array

1. Skriv
   $MyComputers = ”lon-dc1”,”lon-cl1”,”lon-svr1”,”$env:computername” och tryck Enter.

   $MyComputers och tryck Enter.

2. Nu visade vi bara att vi hade tre datorer i arrayen.

3. Med detta kan vi t ex testa Connection mot 4 datorer efter varandra med hjälp av följande:

   Test-Connection -ComputerName $MyComputers

4. Det kommer att bli fel på de servrar som inte är igång.

Slut på Övning 8!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrafrågorna nedan.

Extrauppgifter

1. När du använder **Test-Connection** kan du med hjälp av en parameter ändra så att den bara testar 1 gång mot varje server/dator istället för 4 som standard?

Övning 9 – Pipeline

Tid: ca 10 minuter

I den här övningen tittar vi på Pipeline.

Får du problem med något så fråga så går vi igenom det.

Då labbmiljön inte är helt uppdaterad så kommer vi att använda information från en sparad **Get-HotFix** för att ha lite mer information att jobba med.

Så vi börjar med att kopiera filen hotfixes.xml som finns i katalogen 04_Filer i labbmaterialet till **C:\Scripts**.

Skriv sedan
$HotFixes = Import-CliXml c:\scripts\hotfixes.xml

Uppgift 1 – Sortera

1. Du har problem med en server efter att uppdateringar lagts på servern.

2. För att felsöka lite så vill du ta fram **När installerades patchar senast?** och **Vem installerade dem?**

3. Använd PowerShell konsolen med Administratörsrättigheter.

4. För att visa alla Hotfixes skriver vi:
   $HotFixes

   I vanliga fall skulle vi ha använt **Get-Hotfix –ComputerName "$env:COMPUTERNAME"**

5. Om vi skriver:
   $HotFixes | Sort-Object –Property InstalledOn

6. Så får vi det sorterat baserat på när det installerades.

7. Skriv:
   $HotFixes | Sort-Object –Property InstalledBy

8. Och nu så ser vi av vem eller vad.

Uppgift 2 – Ta fram medlemmar

1. Det är ju en början, men är verkligen ”Source, Description, HotFixID, InstalledBy och InstalledOn” de värden vi behöver mest för att felsöka.

2. Oftast finns det massor av information som kommer ut när man använder en cmdlet, men endast viss förbestämd information visas om man inte själv väljer vilken eller väljer all.

3. För att se vilken information eller vilka medlemmar som finns för en viss cmdlet så använder man cmdlet:en **Get-Member** eller **gm**.

4. Så för att se vilken information vi vill ha fram för hotfixarna så skriv
   $HotFixes | Get-Member eller

   $HotFixes | GM

Uppgift 3 – Välj vilken information du vill se om varje objekt

1. Nu när vi vet vilka olika medlemmar det finns för den här cmdleten så vet vi kanske mer om vilken info vi vill ha för att kunna felsöka lite bättre.

2. Tyvärr är det inte alltid så att alla namn på medlemmar där vi kan få information säger vad det är. Som i det här fallet så är namn inte namn och caption vad är det?

3. Då kan vi till att börja med ta fram all information genom att skriva
   $HotFixes | Select-Object *

4. Nu får du en massa information. Avbryt om du vill genom att trycka Ctrl-C.

5. Den information som är viktig om vi enbart tittar på en server/maskin i taget är:

   HotfixID
   InstalledOn
   InstalledBy
   Description
   Caption

Skriv:
$HotFixes | Select-Object HotfixID, InstalledOn,InstalledBy,Description,Caption

Helst ska man ha hela raden med de olika pipningarna på samma rad.
Men det går att ha det på en ny rad om man har varje cmdlet på en ny rad och avslutar en rad med ett pipetecken.

Uppgift 4 – Välj vilken information du vill se

1. Nu har vi fått ut massa info om alla Hotfixar.
   Men vi kanske bara vill ut de Hotfixar som är listade som Updates.

2. Skriv:
   $HotFixes |
   Select-Object HotfixID, InstalledOn,InstalledBy,Description,Caption |
   Where-Object -Property Description –eq “Update”

3. Och om vi vill ha reda på vilka som systemet installerat själv så kan vi skriva:
   $HotFixes |
   Select-Object HotfixID, InstalledOn,InstalledBy,Description,Caption |
   Where-Object { $_.Description –eq “Update” –and $_.InstalledBy –Match “System”}

Uppgift 5 – Lista i en tabell

1. När man ändrar om och väljer ut en hel del information så tycker PowerShell automatiskt att det blir för mycket när allt inte kan visas ordentligt. Och visar det som en lista.

2. Men givetvis kan få en tabell i alla fall.
   Skriv **$HotFixes |**
   Select-Object HotfixID, InstalledOn,InstalledBy,Description,Caption |
   Where-Object { $_.Description –eq “Update” –and $_.InstalledBy –Match “System”} | Format-Table

   Det går att få upp all information om man använder **Format-Table –Wrap**.

En fråga är hur mycket information man egentligen vill ha ut på skärmen så man skulle nog skippa caption i början på felsökningen och ta fram det senare vid behov.

Uppgift 6 – Se all information på skärmen

1. För att enkelt se all information på skärmen kan man använda Grid View.

2. **$HotFixes |**
   Select-Object HotfixID, InstalledOn, InstalledBy, Description, Caption |
   Where-Object { $_.Description –eq “Update” –and $_.InstalledBy –Match “System”} | Out-Gridview

Uppgift 7 – Spara information till en csv fil

1. Eller så exporterar man informationen enkelt till en csv fil.
2. **$HotFixes |**
   Select-Object HotfixID, InstalledOn, InstalledBy, Description, Caption |
   Where-Object { $_.Description –eq “Update” –and $_.InstalledBy –Match “System”} | Export-Csv -Path C:\Scripts\MinaHotfixar.csv

Slut på Övning 9!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

Nu har du lärt dig lite om vad man kan göra med Pipeline.
Använd det och kombinera lite olika cmdlets för att få fram egen information.
Prova T ex **Get-Service** eller **Get-Process**, med dem kan du göra på liknande sätt som ovan.

Övning 10 – Formatering

Tid: ca 10 minuter

I den här övningen tittar vi på formatering.

För att formatera utdata så kan man göra på en mängd olika sätt.
Formatering blir ännu viktigare när man skriptar.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Visa endast namn

Hittills när vi tagit reda på information så har vi presenterat information från ett objekt som en lista eller tabell.

För att bara presentera ett filnamn på skärmen så kan vi skriva:

1. **$MyFile = (Get-Item c:\windows\windowsupdate.log).Name**

   Nu får vi fram namnet på filen direkt.
   Genom att skriva parenteser runt det vi vill göra och skriva punkt efter så kan vi ta fram en egenskap eller metod direkt.

   Detta kan vi använda ihop med Write-Output för att få fram informationen på skärmen ihop med annat vi vill förmedla.

2. Skriv:

   Write-Output ”Detta är min fil: $MyFile”

3. Det är snyggare/enklare att se om man använder variabler när man ska presentera information.

   Men vi kunde lika gärna ha skrivit vårt kommando direkt:

   Write-Output ”Detta är min fil: $((Get-Item C:\Windows\Windowsupdate.log).Name)”

   Men observera att jag nu skrivit $() runt kommandot.

   Annars ser inte PowerShell att man vill utföra ett kommando mitt i att man vill skriva ut information också.

Uppgift 2 – Visa specifik information i en tabell

Tidigare så har vi pipat objekt till Select-Object för att välja ut vad vi vill visa i en tabell.

Ju fler steg vi använder när vi pipar, desto mer saker behöver processas.

Så vi kan om vi vill istället styra detta direkt med **Format-Table**.

1. Om vi skulle vilja se alla tjänster på datorn vi är på, men vi vill bara veta namnet och statusen och dessutom i den ordningen.

   Så vi kan skriva:

   Get-Service | Format-Table -Property Name, Status

Uppgift 3 – Visa brett

Om vi skulle vilja visa t ex alla tjänster med Displayname över hela vår PowerShell konsol med 3 kolumner så skriver vi:

1. **Get-Service | Format-Wide -Property DisplayName -Column 3**
2. Ser det okej ut?
3. Om du inte hade PowerShell konsolen maximerad så kanske du ser Displaynamnet på tjänsterna med … på slutet på vissa.

   Detta för att PowerShell känner av att den inte kan visa hela raderna.
4. Maximera PowerShell konsolen.
5. Nu vart det inte okej, utan du måste köra om kommandoraden.
6. Bättre?

Slut på Övning 10!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

Frågor:

När vi jobbar med Format-Table.

1. Hur får vi fram standard informationen?
2. Hur gör vi för att dölja rubrikerna för kolumnerna i tabellen?

Övning 11 – Stränghantering

 **Tid:** ca 10 minuter

I den här övningen tittar vi på stränghantering.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Plocka ut domän ur mailadress

När vi jobbar med strängar så är en vanlig sak Split.
Man splitar en sträng från ett specifikt tecken.

För att få mer hjälp om split så kan man skriva **Get-Help about_split**.

Så om företaget du jobbar på ska byta domännamn från foretag.se/foretag.no/foretag.dk till foretag.com så är det ju bra om vi kan få ut vilka olika domäner som vi använder i våra mailadresser.

1. Skriv **$MyMailAddress = ”nisse.hult@foretag.se”**
2. **$MyDomain = $MyMailAddress.Split(”@”)[1]**
3. **$MyDomain**

Vad hände nu undrar du säkert?

När vi splitar något så skapar PowerShell en array av det.

För att bryta ner det lite så backar vi ett steg.

Skriv:

1. **$MyMailAddress = ”nisse.hult@foretag.se”**

2. **$MyMailAddress = $MyMailAddress.split(”@”)**

3. **$MyMailaddress.GetType()**

Och då blir det en array med saker och för att adressera den andra använder vi [1].

Uppgift 2 – Byta mailadresser

Nu har vi fått fram domänen, men för att byta den enkelt så kan vi göra följande.

Skriv:

1. **$MyMailAddress = ”nisse.hult@foretag.se”**

2. **$MyNewMailAddress = $MyMailAddress.Replace(”foretag.se”,”foretag.com”)**

3. **Write-Output $MyNewMailAddress**

Uppgift 3 – Ändra namn i mailadress till små bokstäver

Nu har vi fått fram domäner och vi har bytt domäner i mailadresser.
En annan sak som man ofta vill ha koll på är stora och små bokstäver.

Skriv:

1. **$MyMailAddress = ”nisse.hulT@foretag.com”**

2. **$MyMailAddress = $MyMailAddress.ToLower()**

3. **Write-Output $MyMailAddress**

Slut på Övning 11!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

Extrauppgift 1 – Ändra namn i mailadress till stor begynnelsebokstav

I föregående uppgift så vart allt smått eller stor del stort.
Men tänk om vi vill ha bokstav 1 i förnamnet stor och bokstav 1 i efternamnet stort och resten smått.

Skriv:

1. **$MyMailAddress = ”nisSe.hulT@foretag.com”**
2. **$MyMailAddress**
3. **$MyMailAddress = $myMailAddress.ToLower()**
4. **$MyMailAddress**
   Då är vi där vi var innan. Allt smått!
5. **$MyName = $myMailAddress.split(”@”)[0]**
6. **$MyName**
7. **$MyDomain = $myMailAddress.split(”@”)[1]**
8. **$MyDomain**
9. **$MyFirstName = $myName.Split(”.”)[0]**
10. **$MyFirstName**
11. **$MySurName = $myName.Split(”.”)[1]**
12. **$MySurName**

    Sådär nu har vi brutit ut för och efternamn.

13. **$MyFirstName = ($MyFirstName.Substring(0,1)).ToUpper() + $MyFirstName.Substring(1)**
14. **$MyFirstName**
15. **$MySurName = ($MySurName.Substring(0,1)).ToUpper() + $MySurName.Substring(1)**
16. **$MySurName**

    Och nu har både för och efternamn det vi vill.

Fråga:

1. Hur gör vi för att få ihop en mailadress efter detta?

Övning 12 – Köra filer och skript

 **Tid:** ca 10 minuter

I den här övningen tittar vi på hur PowerShell hanterar körning av filer och skript.

Får du problem med något så fråga så går vi igenom det.

Börja med att skriva:
Set-ExecutionPolicy -ExecutionPolicy Default

Uppgift 1 – Köra filer

1. Skriv **Get-ChildItem -Path C:\windows | ConvertTo-Html | Out-File -FilePath C:\Scripts\Windows.htm**

2. Nu har vi skapat en html fil med outputen från Windows katalogen.

3. Skriv **Windows.htm** och tryck Enter.

4. Nu får vi upp ett felmeddelande.

5. Skriv nu i stället **.\Windows.htm**

6. Detta är helt normal. Det är en säkerhetsgrej att inte ”by default” köra saker i den katalogen man står i utan att man skriver i hela sökvägen eller ”**.**” som representerar nuvarande katalog.

Uppgift 2 – Köra ett PowerShell skript

Se till att du är i Scriptskatalogen, C:\Scripts.

1. Först skapar vi ett superenkelt PowerShell script genom skriva:
Notepad Startaps.ps1 och tryck Enter.

2. Tryck Yes på frågan om du ska skapa en ny fil.

3. Skriv:

   Cls
   Get-ChildItem -Path C:\windows

4. Spara och stäng Notepad.

5. Skriv **.\Startaps.ps1** och tryck Enter.

6. Nu får vi upp ett felmeddelande o matt vi inte får köra script på maskinen.

7. Detta är det normala (standard) beteendet för PowerShell skripts.

8. För att ändra detta så stäng ner PowerShell.

9. Starta upp en ny PowerShell konsol som Administratör.

10. Skriv **Set-ExecutionPolicy -ExecutionPolicy RemoteSigned** och tryck på Enter.

11. Tryck på Enter när du får upp frågan om du vill ändra policyn.

12. Stäng ner PowerShell konsolen.

13. Starta upp PowerShell konsolen som vi skapade tidigare.

Skriv **.\startaps.ps1** och tryck Enter.

Slut på Övning 12!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

Först skapar vi ett superenkelt VBScript script genom skriva
notepad startavbs.vbs och tryck Enter.

1. Tryck Yes på frågan om du ska skapa en ny fil.

2. Skriv:

   Wscript.Echo ”Hej 01”
   Wscript.Echo ”Hej 02”
   Wscript.Echo ”Hej 03”

3. Spara och stäng Notepad.

4. Skriv **.\startavbs.vbs** och tryck Enter.

5. Nu kommer du att få upp popupmeddelanden x 3 som du måste stänga genom att klicka på OK.

6. **Fråga:**
   Varför vart det såhär?

7. **Svar:**
   Som standard är det wscript.exe som kör VBScript filer i Windows om man bara kör dem utan att skriva cscript.exe innan och Wscripts beteende är sådär.

8. Om man vill testa själv så fortsätt, annars kan du sluta här.

9. Starta upp en PowerShell konsol som Administratör.

10. Skriv **Wscript //H:Cscript** och tryck Enter.

11. Skriv **.\startavbs.vbs**

12. Nu får du alla meddelanden i PowerShell konsolen.

13. Skriv **Wscript //H:Wscript** och tryck Enter.

Övning 13 – Executionpolicy

Tid: ca 10 minuter

I den här övningen tittar vi på hur PowerShells Executionpolicy.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Nuvarande

1. Det är ganska ofta man råkar på att ett skript inte körs på en dator och man ser oftast ganska fort om det beror på execution policyn.

   Dock så ser man inte vilken direkt.

2. För att lista execution policyn på datorn så skriver du ”**Get-Executionpolicy**”.
3. För att lista alla på datorn så skriver du ”**Get-Executionpolicy -List**”.

Uppgift 2 – Ändra

1. Ändra execution policy till **RemoteSigned**

2. Använd hjälpen för **Set-ExecutionPolicy** om du behöver hjälp.

3. Om du får ett felmeddelande så tänk på administratör…

Slut på Övning 13!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

1. Surfa till https://gallery.technet.microsoft.com/scriptcenter och sök på ”Create real fake names”.

2. Ladda hem skriptet och lägg det i **c:\scripts** katalogen.

3. Kör skriptet

Frågor:

1. Varför får vi inte köra detta skript?
2. Hur skulle vi kunna göra för att få köra detta skript?

   TIPS!
   Det finns minst 2 sätt!

Övning 14 – PowerShell konsolen vs Skript Editorer

Tid: ca 10 minuter

I den här övningen tittar vi på hurVisual Studio Code fungerar.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Installera Visual Studio Code

Till skillnad från PowerShell ISE så måste vi installera Visual Studio Code innan vi kan börja använda den.

Visual Studio Code finns att hämta hem här:
http://code.visualstudio.com

Klicka på **Download for Windows**.

Installera som standard men välj högerklicka i explorer.

Uppgift 2 – Ställ in så att PowerShell blir default språk

För att alltid få Visual Studio Code att starta upp med PowerShell så måste vi göra lite inställningar och installera PowerShell extension.

1. Starta Visual Studio Code.

2. I menyn till vänster med olika ikoner så finns det en med 4 kuber.
   Det bör vara den längst ner.

3. Klicka på den.

4. Överst finns det en sökruta.
   Sök på PowerShell.

5. Klicka på den översta som dyker upp.

6. Klicka på install.

7. Nu har vi installerat stöd för PowerShell.

8. För att nya filer ska starta upp med PowerShellstöd från början så behöver vi välja att PowerShell ska vara defaltspråk.

9. Klicka på Shift + Ctrl + P

10. Skriv Open User Settings i sökrutan.

11. Klicka på översta som dyker upp.

12. I sökrutan skriver du default language.

13. Under **Files: Dafault Language** skriver du **powershell**

14. Stäng ner Settings på krysset högst upp bredvid Settings.

Uppgift 3 – Autocomplete

1. Öppna upp Visual Studio Code.
2. Klicka på **File** i menyn och **New File**.
3. Skriv **Get-**
4. Nu får vi hjälp av IntelliSense.
5. Antingen väljer vi något i menyn eller så vill vi använda TAB för att kunna blädra bland cmdlets.
6. Fortsätt att skriva och skriv **Get-Childitem**.
   Testa även att skriva annat för att se hur listan funkar.
7. Om du nu testar TAB vad händer?
8. Visual Studio Code har inte TAB completion påslaget som standard.
9. Tryck Shift + Ctrl + P
10. Skriv Open User Settings i sökrutan.
11. Klicka på översta som dyker upp.
12. I sökrutan skriver du tab completion.
13. Under **Editor: Tab Completion** ändrar du **Off** till **On**.
14. Stäng ner Settings på krysset högst upp bredvid Settings.
15. Om du går tillbaka till där du skrev **Get-**
16. Prova **TAB** nu. I värsta fall ta bort **Get-** och skriv det igen och testa TAB nu.

Uppgift 4 – Kör Script

1. För att köra ett skript i Visual Studio Code eller i de flesta andra PowerShell editorer som finns så trycker man på **F5**.
2. Tryck på **F5**
3. Om du inte kan köra ditt kommando nu så beror det nog på att du inte ändrat execution policy till RemoteSign.

Uppgift 5 – Kör endast en rad i ett script

1. Om vi nu skriver **Get-Date** på raden under **Get-ChildItem** och trycker **F5**.

2. Nu så körs båda raderna och all information susar förbi nere i Console fönstret.

3. För att bara köra den sista raden av kod så markerar vi **Get-Date** och trycker **F8**.

Uppgift 6 – Konsolfönstret

1. I första delen av denna övning så skrev vi in det vi ville i övre fönstret (Skript fönstret). Men vill man bara testa cmdlets så kan man även använda konsolfönstret (det nedre fönstret, Terminal) direkt.

2. Prova att skriva **Get-ChildItem** och tryck **Enter**.

Uppgift 7 – Ändra teman

Även i Visual Studio Code kan man ändra färg.

Här finns det färdiga teman också.

1. Klicka på **kugghjulet (manage)** nere till vänster**.**

2. Välj **Color Theme**.

3. Det finns ett gäng som standard + ett par till när man installerat PowerShell extension.

4. Duger inte dessa så kan man klicka på Install additional color themes.

5. Och så finns det en lång lista på teman att välja på.

Slut på Övning 14!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 15 – Skript

Tid: ca 15 minuter

I den här övningen tittar vi på hur PowerShell skript är uppbyggda.

Får du problem med något så fråga så går vi igenom det.

Uppgift 1 – Första skriptet

1. För att skapa ett första skript så startar vi upp Visual Studio Code om den inte är igång sedan tidigare.

2. Gå antingen på **File** menyn och välj **New File** eller tryck **Ctrl + N**.

3. Skriv följande:

   Clear-Host
   $MinDator = $env:COMPUTERNAME
   Write-Output $MinDator

4. Tryck **F5** för att köra det

5. Tänk på att spara det du skapar så ofta som möjligt.

6. Tryck antingen **Ctrl-S** eller gå upp på **File** Menyn och välj save.

7. Spara under **C:\scripts**

8. Vad gör raderna i skriptet?

Uppgift 2 – Kommentarer

1. Kommentarer är viktiga i skript.
   Annars glömmer vi vad vi har gjort eller andra vet inte hur vi tänkte.

2. För att kommentera en rad i PowerShell så skriver man ett brädgårdstecken.

3. Så första kommenteringen blir alltså ovanför **Clear-Host**.
   Skriv:
   # Rensa skärmen

4. Gör samma sak med alla rader.

5. När vi senare börjar skriva längre skript så behöver vi inte kommentera varje rad.
   Utan man brukar rikta in sig på att kommentera stycken eller viktiga delar.

Uppgift 3 – Information i början

1. Att kommentera saker i själva skriptet är viktigt.
   Dessutom är det viktigt att kommentera i början på skriptet för att alla ska veta vem som gjort skriptet och vad det är till för.

2. En bra regel är att använda dessa kommentarer/kommentarsbaserad hjälp:

```
<#
.SYNOPSIS
        Detta skript används för att
.DESCRIPTION
        Detta skript används för att
.NOTES
                        Author:     Fredrik Wall
                        Email:      fredrik.wall@retune.se
                        Company:    Retune AB
                        Created:    2018-03-03
                        Updated:    2018-04-03
#>
```

3. Förutom att det blir lätt att se information om skriptet så får man hjälp till sitt skript på köpet.

4. Skapa ett nytt skript och lägg in kommentarsbaserad hjälp som ovan.

5. För att slippa att skriva in allt så kan du kopiera innehållet från filen kommentarsbaserad_hjälp.txt som ligger i den utdelade katalogen under 03_Demon\15_Skript.

6. Spara skriptet som **kommentarsbaseradhjalp.ps1 under c:\scripts**.

7. Gå ner till kommandofönstret, ställ dig i rätt katalog och skriv
   Get-Help .\kommentarsbaseradhjalp.ps1 -Full

Uppgift 4 – Hur körs ett PowerShell skript?

1. Skapa ett nytt skript

2. Skriv följande:

   Clear-Host
   Write-Output $MinVariabel
   $MinVariabel = "Hej"

3. Kör skriptet

4. Vad händer?

5. Kör skriptet igen

6. Varför funkar det nu?

Slut på Övning 15!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

1. Om vi skriver ett skript som t ex är beroende av en viss version av PowerShell så kan vi styra detta på ett enkelt sätt.

2. Skapa ett nytt skript.

3. Skriv:

4. **#Requires -Version 7.0**

5. På första raden

6. Skriv sedan:

   Write-Output ”Hejsan!”

7. Spara skriptet

8. Kör skriptet.

9. Vad hände?

10. Ändra översta raden till:

    #Requires -Version 4.0

11. Kör skriptet igen

Övning 16 – Villkor

Tid: ca 10 minuter

I den här övningen tittar vi på villkor.

Får du problem med något så fråga så går vi igenom det.

När vi skriptar så ställer vi ofta olika villkor.

Uppgift 1 – Om något är mer än något

1. Skapa ett nytt skript.
2. Skriv **10 -gt 5** i editeringsfönstret och tryck F5.
3. Om vi skulle vilja omvandla detta till mer praktiskt så kan vi skriva följande:
4. **$MyFiles = Get-ChildItem -Path C:\scripts**

```
if ($myFiles.Length -gt 2) {
    Write-Output “Dax att rensa filer”
}
```

Uppgift 2 – Om något är sant gör detta

Vi kan skapa variabler som är true eller false och sedan använda det för att t ex visa mer information.

1. Skapa ett nytt skript
2. Skriv:

```
$Debug = $true

if ($Debug -eq $true) {
   Write-Output “Dagens datum är:”
}

Get-Date
```

3. Kör skriptet
4. Ändra **$Debug = $true** till **$Debug = $false**

5. Kör skriptet

Slut på Övning 16!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 17 – Loopar

 **Tid:** ca 10 minuter

I den här övningen tittar vi på loopar.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Gör detta 100 gånger

1. Öppna upp Visual Studio Codesom administratör.

2. Gå till Menyn och File -> New eller Tryck Ctrl + N

3. Nu kommer det upp en ny skriptruta.

4. Spara den under c:\scripts som loopar01.ps1

5. Skriv **1..100 | foreach-object { calc }**

6. Upps vad gjorde du nu?
   Startade 100 miniräknare ☺

   Fungerade det inte med calc, prova att skriva **notepad** istället för **calc**.

7. Lugn de får du bort genom att skriva **Stop-Process –Name calc*** i det nedersta fönstret.

   Har du startat notepad 100 gånger istället för calc. Skriv notepad istället för calc i Stop-Process.

8. Om vi gör om samma sak men skriver:

```
1..100 | Foreach-object {
        Write-Host ”Startar Notepad nr $_”
        notepad
}
```

   Så ser vi även i outputdelen i mittendelen vad vi gör.

9. För att snygga till skriptet en aning för att få det extra lättläst så markera rad 2 och 3 och tryck TAB.

10. Det vi gör att vi tar 1 till 100 som objekt och kör **Foreach-Object** så att vi per objekt kan utföra saker.

    Så inom **{** och **}** utför vi allt för varje objekt.

    I detta fall skriver vi bara ut vad vi gör först och använder $_ för att tala om vilket objekt vi använder.

    Och sedan startar vi Notepad.

    Och detta görs alltså 1 till 100 gånger.

11. Spara skriptet.

Uppgift 2 – Gör detta för varje rad i en textfil

1. Skriv följande I terminalen:
   Set-Content -Value "lon-dc1","lon-srv1","lon-cl1" -Path C:\Scripts\Serverlist.txt

2. Skapa ett nytt skript i Visual Studio Code editorn och döp det till S**erverlist.ps1**.

3. Skriv:

```
$MyServerList = Get-Content C:\Scripts\Serverlist.txt

Foreach ($Server in $MyServerList) {
        Write-Output $Server
}
```

4. Tryck **F5**

5. Nu får vi ut en lista på de servrar vi har i vår server fil.

6. **Foreach** läser helt enkelt in varje rad som **$Server** ifrån vår fil som vi lagrat i **$MyServerList**.

7. Och i detta fall är varje rad en server eller dator.

Slut på Övning 17!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

1. Om vi har ett skript där vi vill att användaren ska skriva in t ex ett servernamn så kan vi använda **do** och **while** för att skriptet ska vänta tills användaren fyllt i ett rätt namn eller åtminstone åt rätt håll.

2. Skapa ett nytt skript i PowerShell ISE och spara det till do01.ps1
3. Skriv:
   do {

   $Input = Read-Host "Servernamn: "
   } while (!($Input -like "lon*"))

4. Tryck F5

5. Skriv **lab-srv01** som input.

6. Vad händer och varför?

7. Om du skriver **lon-dc01** istället vad händer då?

8. Till detta kan man lägga vad som helst efteråt, så när skriptet går vidare fångar man upp det användaren skrev med detta:

   Get-process -ComputerName $input

Övning 18 – Funktioner

Tid: ca 10 minuter

I den här övningen tittar vi på funktioner.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Min första funktion

Vår första funktion är en enkel funktion som rensar temp katalogen för den användaren den körs för lokalt på maskinen där skriptet körs.

1. Skapa en ett nytt skript i PowerShell ISE och spara det som funktioner01.ps1

2. Skriv:

```
function Lista-Filer {

        Get-Childitem -Path c:\Scripts -Recurse
}
```

3. Tryck F5

4. Nu händer det inget

5. För att köra/anropa en funktion så måste vi anropa den

6. Skriv **Lista-Filer** under sista måsvingen ”}”

7. Tryck F5

Uppgift 2 – Lägga till lite intelligens

Om vi ändrar om funktionen till detta:

```
function Lista-Filer {
	Param($Katalog)
	Get-Childitem -Path $Katalog
}

Clear-Host

Write-Output "Ett"
Lista-Filer -Katalog "C:\Scripts"

Write-Output "Två"
Lista-Filer -Katalog "C:\Windows"
```

Så ser vi att vi kan lägga till i princip vad som helst i funktionen och anropa den med ett katalognamn för att göra något.

I detta exempel byter vi helt enkelt ut den manuellt skriva sökvägen till den katalog där vi vill titta på filer till variabeln $katalog som vi sätter via en parameter.

Slut på Övning 18!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 19 – Text

Tid: ca 10 minuter

I den här övningen tittar vi på texthantering.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Räkna rader i en textfil

Entext.txt finns i kursmaterialet i katalogen **04_Filer**.
Kopiera den till **C:\Scripts**.

Ibland kan man vilja veta hur många rader det är i en textfil.
Detta är enkelt.

Vi använder oss av cmdleten **Get-Content** som hämtar innehåll från filen och så anropar vi egenskapen count för att få reda på hur många rader filen innehåller.

(Get-Content C:\Scripts\Entext.txt).Count

Dock har vi nu räknat alla rader även tomma rader.

För att räkna endast rader med text så skriver vi följande:

Get-Content C:\Scripts\Entext.txt | Measure-object -Line

Uppgift 2 – Hitta ord i en textfil

För att få PowerShell att hitta ett ord i en textfil kan vi på enklast möjliga sätt använda **Select-String** och skriva:

1. **Get-Content C:\Scripts\Entext.txt | Select-String ”kurs”**
2. Nu kommer det inte upp något
3. Om vi ändrar **kurs** till **no** så ska vi få träff.

Frågor:

1. Är Select-String ”Case sensitive”?
2. Hur gör vi den Case Sensitive?

Uppgift 3 – Hitta ord i fler textfiler

I föregående uppgift så tog vi en specifik fil och letade efter ett ord.
Vi kan använda liknande metod men i en hel katalog eller på hela hårddisken.

Då skriver vi istället:

Get-ChildItem -Path C:\ -Recurse | Select-String -Pattern "Select"

Uppgift 4 – Ändra text i en textfil

Skriv följande i PowerShell ISE:

Set-Content -Value ”Detta är en text som inte säger något, men som innehåller lite text utan egentligt innehåll och bara text” -Encoding utf8 -Path C:\Scripts\litetext.txt

Och kör raden med F8.

Skriv sedan nedanstående i PowerShell ISE och kör det med F8.

(Get-Content C:\scripts\litetext.txt).Replace('text', 'humor') | Set-Content C:\Scripts\litetext.txt

Slut på Övning 19!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 20 – Regular Expressions

Tid: ca 5 minuter

I den här övningen tittar vi lite snabbt på Regular Expression.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Hitta personnummer i en text

Vi kan använda -match för att hitta mönster med RegEx

$Text = 'Mitt personnummer är 940203-0089'
$Text -match '\d\d\d\d\d\d-\d\d\d\d'

Om vi får en träff så får vi True annars False

Uppgift 2 – Hitta personnummer i filer

Vi kan använda Select-String -Pattern för att hitta mönster med RegEx.

Get-ChildItem -Path c:\ -Recurse | Select-String -Pattern '\d\d\d\d\d\d-\d\d\d\d'

Uppgift 3 – Maskera personnummer

Även replace klarar av RegEx.

$Meddelande = "Mitt personnummer är 950212-0034"
$Meddelande -replace '\d\d\d\d\d\d-\d\d\d\d', '######-####'

Slut på Övning 20!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 21 – Filsystemet

Tid: ca 15 minuter

I den här övningen tittar vi på filhantering.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Hitta filer

1. Du vill leta upp alla filer lokalt på en hårddisk som slutar med *.txt.
2. Skriv **Get-ChildItem -Path C:\ –Filter *.txt –Recurse**
3. Nu får du upp info om alla **txt** filer på **C:**

Frågor:

1. Om du skulle vilja slippa alla felmeddelande där du inte har rättigheter att leta hur skulle då kommandot bli? Tänkt till, titta på tidigare uppgifter eller fråga!

2. För att välja ut vilken information du vill se för att slippa se allt hur gör du då?

3. Vilken information skulle vara relevant?

4. Om man vill hitta en specifik fil hur gör man då?

Uppgift 2 – Skapa

1. Skapa en katalog under **C:\Scripts** som heter lab.

2. Skriv **New-Item -Path C:\Scripts -Name Lab -ItemType Directory**

3. Skapa en katalog under **c:\Scripts\Lab** som heter **Backup** på samma sätt som ovan.

4. Skriv **New-Item -Path C:\Scripts\Lab -Name Backup -ItemType Directory**

Uppgift 3 – Kopiera filer

1. Skapa en fil genom att skriva:
   New-Item -Path C:\Scripts -Name do01.ps1 -Value "Detta är en dummy file"

2. Kopiera enstaka filer gör du genom att använda
   Copy-Item -Path C:\Scripts\do01.ps1 -Destination c:\Scripts\Lab\Backup

3. Nu när vi vet hur vi ska hitta och kopiera filer så kan vi kopiera dem till vår backup katalog

4. Ett enkelt sätt är:
   Get-Childitem -Path C:\Scripts -Filter *.ps1 -ErrorAction SilentlyContinue | copy-item -Destination C:\Scripts\Lab\Backup

Uppgift 4 - Ta bort filer

1. Nu vill du inte längre ha kvar dina filer och vill ta bort alla filer i backup katalogen.

2. **Remove-Item -Path C:\Scripts\Lab\Backup*.***

Uppgift 5 – Flytta filer

1. För att istället flytta filer så använder du Move-Item.

2. **Move-Item -Path C:\Scripts\do01.ps1 -Destination C:\Scripts\Lab**

3. Vill du vara säker på att den hamnade där kan du använda cmdleten **Test-Path -Path C:\Scripts\Lab\do01.ps1**.

Uppgift 6 – Ändra namn på filer

För att döpa om en fil använder vi Rename-Item.

Skriv följande på samma rad:
Rename-Item -Path C:\Scripts\minakataloger.txt -NewName C:\Scripts\minakataloger_ny.txt

Slut på Övning 21!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

Extrauppgift 1 – Hitta filer och kataloger skapade detta år

För att lista alla filer i Windows katalogen som har CreationTime från 2019-01-01 till nu så kan man göra följande:

Skapa ett nytt skript med följande.

```
$MyFiles = Get-ChildItem -Path 'C:\Windows'
$MyYearFiles = $MyFiles | Where-Object -Property CreationTime -ge "2019-01-01"
$MyYearFiles |Select-Object FullName,CreationTime | Sort-Object -Property CreationTime
```

Dessutom så sorterade vi så man kan se äldsta först.

Extrauppgift 2 – Lista alla logfiler i windowskatalogen

Skapa ett nytt skript.

Om man vill hitta alla logfiler i en katalog så gör man bara enligt nedan:

Get-ChildItem -Path C:\windows -Filter *.log

För att filtrera ut fler filändelser på samma gång så är det enklast att göra följande:

```
Get-ChildItem -Path C:\Windows |
Where-Object { $_.extension -eq ".log" -or $_.extension -eq ".exe" }
```

Extrauppgift 3 – Tomma filer

För att hitta tomma filer så gör vi nästan på samma sätt:

```
Get-ChildItem -Path C:\Windows -Recurse |
Where-Object -Property length -eq 0
```

Övning 22 – XML, INI och andra inställningsfiler

Tid: ca 15 minuter

I den här övningen tittar vi på hur vi kan använda inställningsfiler.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Läsa en XML fil

I kursmaterialet i katalogen **04_Filer** finns filen **template_xml.xml**.
Kopiera den till katalogen **c:\scripts** och döp om den till **myxml.xml**.

Skapa följande skript:

```
$myXMLFile = "$PSScriptRoot\myxml.xml"
$xml = New-Object -TypeName XML
$xml.Load($myXMLFile)
$Xml.Settings.Setting
```

Nu får vi fram informationen från XML filen.

Om vi lägger till en section med setting i myxml.xml enligt nedan:

```
<?xml version="1.0"?>
<Settings>
    <Setting>
    <Company>Company 01</Company>
    <Location>Stockholm</Location>
    <FileServer>Filesrv 01</FileServer>
    </Setting>
    <Setting>
    <Company>Company 02</Company>
    <Location>Sundsvall</Location>
    <FileServer>Filesrv 02</FileServer>
    </Setting>
</Settings>
```

Kör vi skriptet igen så får vi ut mer information och fler företag.
Vi kan ändra företag till t ex Avdelning.

Nu får vi ut all information ifrån xml filen.

Uppgift 2 – Läsa en PSD1 fil

PowerShell har datafiler som man enkelt kan skapa och använda som inställningsfiler.

I kursmaterialet i katalogen **04_Filer** finns filen **mypsdatafile.psd1**.
Kopiera den till katalogen **c:\scripts**

Den innehåller följande:

```
@{
        Company= "My Company"
        Location = "My City"
        MyDC=”lon-dc1”
}
```

Sedan skapar du ett nytt skript med följande kod:

```
$MyPsdFile = "$PSScriptRoot\ mypsdatafile.psd1"
$MySettings = Import-PowerShellDataFile -Path $MyPsdFile

$MySettings
```

Syftet med att använda settingsfiler på olika sätt är för att slippa hårdkoda saker i ett skript.

Man läser in t ex, logfilskatalog, företagsnamn, servernamn från en settingsfil och knyter till variabler i skriptet som man sedan använder på rätt ställen.

Så om vi skriver följande i ett skript.

```
$MyPsdFile = "$PSScriptRoot\mypsdatafile.psd1"
$MySettings = Import-PowerShellDataFile -Path $MyPsdFile

$MyDC = $MySettings.MyDC

Test-Connection $MyDC
```

Slut på Övning 22!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Om du har kommit hit och känner att du har tid över eller vill ha lite extra efter kursen så kan du ta en titt på extrauppgifterna nedan.

Extrauppgifter

Extrauppgift 1 – XML fortsättning

Skulle vi bara vilja ha ut information om t ex Company 01 så kan vi skriva följande skript:

```
Clear-Host

$myCompany = "Company 01"

$myXMLFile = "$PSScriptRoot\myxml.xml"
$xml = New-Object -TypeName XML
$xml.Load($myXMLFile)
$mySetting = $Xml.Settings.Setting

foreach ($setting in $mySetting) {
   if ($myCompany -eq $setting.Company) {
      $companyName = $setting.Company
      $companyLocation = $setting.Location
      $companyFileServer = $setting.FileServer
   }
}

Write-Output "Company: $companyName`nLocation: $companyLocation`nFile Server: $companyFileServer"
```

Endast fantasin sätter gränser här.

Vi kan lagra information om t ex applikationer och så skapa en funktion som vi kan anropa för att t ex installera, hämta eller konfigurera applikationer.

Extrauppgift 2 – Läsa en INI fil

I kursmaterialet i katalogen **04_Filer** finns filen **template_ini.ini**.
Kopiera den till katalogen **c:\scripts** och döp om den till **myini.ini**.

Att läsa värden ur en INI fil om de bara finns en gång är lätt.
Men om det finns samma saker under fler kategorier så blir det klurigare.

Så för att göra det enkelt för sig så kan man installera modulen **PSIni**.

Skapa ett nytt skript och skriv följande:

```
Install-Module psini
Import-Module psini

$FileContent = Get-IniContent "C:\Scripts\myini.ini"

$FileContent["Company"]["Name"]
```

Inifilen läses in I en ordedDictonary och kan därför adresseras med hjälp av ovanstående. Mer om detta finns i about Hash Tables.

För att skriva till inifilen gör man så här:

```
Set-IniContent -FilePath "C:\scripts\myini.ini" -Sections 'Company' -NameValuePairs @{'Name'='Company 01' ; 'Location' = 'City 01'}  | Out-IniFile "C:\scripts\myini.ini" -Force
```

Övning 23 – Felhantering

 **Tid:** ca 10 minuter

I den här övningen tittar vi på felhantering.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Stoppa vid fel

För att göra en enkel felhantering som stannar vid fel så använder vi Try and Catch.

```
Try
{
   $myFile = Get-Content c:\scripts\notmyfile.txt -ErrorAction Stop
   Write-Host "Det funkade!"
}
Catch
{

   Write-Output "Något gick fel!"
   Break
}
  Write-Output ”Slut på skriptet!”
```

Uppgift 2 – Strunta i felet fortsätt

Ibland vill vi bara strunta i fel, kanske logga dem. Men inte stanna skriptet.

Då gör vi följande:

```
Try
{
   $myFile = Get-Content c:\scripts\notmyfile.txt -ErrorAction Stop
   Write-Host "Det funkade!"
}
Catch
{

   Write-Output "Något gick fel!"
}
Write-Output ”Slut på skriptet!”
```

Uppgift 3 – Visa felmeddelanden

Hittils har vi bara stoppat och visat eget meddelande.
För att visa vad PowerShell tycker är fel så kan vi Exeption.

Skriv:

```
Try
{
   $myFile = Get-Content c:\scripts\myfile.txt -ErrorAction Stop
   Write-Host "Det funkade!"
}
Catch
{

   Write-Output "Något gick fel!"
   Write-Output "Felmeddelande: $($_.Exception.Message)"
   Break
}

Write-Output "Slutet"
```

Slut på Övning 23!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 24 – Registret

Tid: ca 10 minuter

I den här övningen tittar vi på hur vi kan hantera registret.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Läsa ut alla Uninstall strängar från registret

1. Du vill ta reda på några av de Uninstall nycklar som finns på din maskin.
   Dessa finns i registret och tas enklas fram med:
   Get-ChildItem HKLM:\Software\Microsoft\Windows\Currentversion\Uninstall | Get-ItemProperty | select DisplayName,UninstallString

Uppgift 2 – Skapa en registernyckel för företaget

1. Ibland vill man han en registernyckel för företaget under **HKEY_LOCAL_MACHINE\Software\FöretagetsNamn** där man sparar register värden.
   För t ex versionen på installerade OS Imagen. Om man har gjort någon ändring etc.

2. Skapa själva nyckeln genom att använda New-Item.
   New-Item -Path HKLM:\Software -Name MyCompany

3. För att sedan skriva in själva värdet så använder vi New-Itemproperty.
   New-Itemproperty -path HKLM:\Software\MyCompany -name OSImage -value “1.02” –force
   New-Itemproperty -path HKLM:\Software\MyCompany -name CitrixUpgrade -value “OK” –force

Uppgift 3 – Läsa ett värde ur registret

För att enkelt kunna läsa ut all info under en registernyckel så skriver man följande:

$MyKey = Get-Item -path HKLM:\SOFTWARE\MyCompany
Write-Output $MyKey

Och vill man ha ett specifikt värde så kan man skriva:

$MyKey = Get-ItemProperty -Path HKLM:\SOFTWARE\MyCompany
Write-Output $MyKey.OSImage

Uppgift 4 – Ändra ett värde i registret

Och vill vi ändra värdet på OSImage så skriver vi bara:

Set-ItemProperty -Path HKLM:\SOFTWARE\MyCompany -name OSImage -Value "2.0"

Slut på Övning 24!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 25 – Processer

Tid: ca 10 minuter

I den här övningen tittar vi på hur vi kan hantera processer.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Lista processer

1. För att lista alla processer använder vi **Get-Process**.

2. För att sortera dem via CPU belastning
   Get-Process | sort CPU –Descending

3. För att visa endast de 10 mest CPU krävande processerna
   Get-Process | sort CPU –Descending | select –First 10

Uppgift 2 – Lista enbart en eller vissa processer

1. Detta gör man enklast genom att bara skriva t ex **Get-process notepad**

2. Hur gör du för att lista alla processer som startar med s?

Uppgift 4 – Starta notepad

1. Starta notepad med hjälp av **Start-Process**
2. Skriv **Start-Process Notepad**

Uppgift 5 – Stäng alla notepad

För att enklast stänga ner notepad gör man följande:

Get-Process notepad | Stop-Process

Uppgift 6 – Hur många processer är igång?

Om man tycker att en dator eller server går långsamt så kanske man vill kolla antal processer som är igång.

(Get-Process).Count

Vi kör Get-Process med propertyn Count.

Slut på Övning 25!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 26 – Tjänster

Tid: ca 10 minuter

I den här övningen tittar vi på hur vi kan hantera tjänster.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Lista tjänster

För att lista alla tjänster använder vi:

Get-Service

Uppgift 2 – Lista en tjänst

För att lista 1 tjänst så lägger vi till vilken tjänst vi vill titta på.

Get-Service -Name Bits

Uppgift 3 – Lista mer information om en tjänst

Get-Service -Name Bits | Select *

Uppgift 4 – Lista Stoppade tjänster

För att lista alla stoppade tjänster så använder vi enklast Where-object och använder -property Status för att ta status som är -eq (lika med) Stopped. Och snygga till tabellen lite med AutoSize.

Get-Service | Where-Object -property Status -eq "Stopped" | Format-Table -AutoSize

Uppgift 5 – Starta, Stoppa och Restarta en tjänst

För att Stoppa en tjänst skriver vi:

Start-Service -Name Bits

För att starta en tjänst skriver vi:

Stop-Service -Name Bits

För att restarta en tjänst skriver vi:

Restart-Service -Name Bits

Slut på Övning 26!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 27 – Eventloggen

 **Tid:** ca 10 minuter

I den här övningen tittar vi på hur vi kan hantera Eventloggen.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Lista vilka loggar som finns

Det finns ett gäng eventloggar på en dator och förhoppningsvis fler på en server.
Så att lista dessa är ju ett måste om vi ska börja jobba med eventloggar.

Get-EventLog -list

Och vill vi göra detta mot en annan dator så kan vi göra följande:

Get-EventLog -list -ComputerName %datornamnet%

Uppgift 2 – Lista event i en eventlogg

Vill vi visa alla poster i Applikationslogen så skriver vi följande:

Get-EventLog -LogName Application

Uppgift 3 – Lista en specifik typ av meddelande

För att visa en typ av poster så kan man använda EntryType.
Skriv nedanstående och använd TAB på EntryType för att se vilka olika typer det finns.

Get-EventLog -LogName Application -EntryType <TAB>

Slut på Övning 27!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 28 – WMI

Tid: ca 10 minuter

I den här övningen tittar vi på hur vi kan hantera WMI

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Lista WMI klasser

För att lista alla WMI klasser som finns på en dator så skriver man:

Get-WMIObject -List

Och för att använda CIM istället och lista alla tillgängliga CIM klasser så skriver vi:

Get-CIMClass

Uppgift 2 – Ta reda på information om Operativsystemet

En vanlig sak att ta reda på är information om Operativsystem och färdiga skript innehåller ofta WMI frågor.

För att ta reda på all information som finns om operativsystemet via WMI på datorn så skriver vi:

Get-WMIObject -Class ”Win32_operatingSystem” | Select *

Och för att istället använda CIM så skriver vi:

Get-CimInstance -ClassName "Win32_OperatingSystem" | Select *

Uppgift 6 – Ta reda på serienumret

En annan vanlig sak man vill ta reda på är serienumret.

Det får vi fram genom att skriva:
(Get-CimInstance -ClassName "Win32_Bios").SerialNumber

Slut på Övning 28!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 29 – Användning av moduler

 **Tid:** ca 10 minuter

I den här övningen tittar vi på hur vi kan hantera moduler.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Lista moduler som är laddade i sessionen

För att lista moduler som är laddade i den nuvarande sessionen så skriver man:

Get-Module -All

Uppgift 2 – Lista alla moduler som är tillgängliga på datorn

För att lista alla moduler som finns tillgängliga på datorn så skriver man:

Get-Module -ListAvailable

Dock visas bara de moduler som på ett korrekt sätt installerats på datorn.
Moduler som importeras direkt från en .psd1 fil syns inte här.

Uppgift 3 – Lista moduler från PSGallery

Numera kan man även installera och leta efter moduler direkt från PSGallery.

Find-Module -Repository PSGallery

Uppgift 4 – Installera PSWindowsUpdate

En av mina favorit moduler heter PSWindowsUpdate och finns på PSGallery.

Find-Module PSWindowsUpdate | Install-Module -force

Uppgift 5 – Använd PSWindowsUpdate

Och för att använda den så skriver man bara.

Import-Module PSWindowsUpdate

För att hitta kommandona för modulen så skriver vi:

Get-Command -Module pswindowsupdate

För att se uppdateringshistoriken skriver vi:

Get-WUHistory

Slut på Övning 29!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 30 – Skripta i företagsmiljöer

Tid: ca 5 minuter

I den här övningen går vi igenom att skripta i företagsmiljöer.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Strukturera med namnstandard

När vi skriptar på ett företag.
Så bör vi vara noga med struktur och namnstandard.

Gå igenom de skript du har skapat nu.

Vad heter de?

Döp om så att de heter något som är relevant för skriptet.

T ex **Get-HotfixesFromAllServers.ps1, Get-PingStatus.ps1**

Uppgift 2 – Kommentera mera

Samma sak här.
Se till att skripten innehåller kommentarsbaserad hjälp och kommentarer där det behövs.

Uppgift 3 – Inget hårdkodat

Så lite som möjligt ska vara hårdkodat.

Parametrar på skript är bra eller att läsa från inställningsfil.
Använd variabler tidigt i skripten.

Gå igenom det du gjort och försök tänka till hur du skulle kunna göra.

Slut på Övning 30!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 31 – PowerShell profilen

 **Tid:** ca 5 minuter

I den här övningen tittar vi på PowerShell profilen.

Får du problem med något så fråga så går vi igenom det

Uppgift 1 – Skapa en Profil

1. Skriv ”**$profile**”.

2. Nu får du sökvägen till var din användares PowerShell profil för Windows PowerShell konsolen ska vara.

3. Skapa denna genom att skriva ”**New-Item -Path $profile -ItemType file -force**”.

4. Och för att editera den så skriver vi ”**notepad $profile**”.

5. Här kan man lägga in sina egna Aliases, Funktioner, ändra promptens utseende m.m m.m
6.
7. Skriv:

   Write-Output ”Välkommen till min PowerShell”

Uppgift 2 – Startkatalog

Om vi vill ha samma startkatalog oavsett om vi startar PowerShell som RunAsAdministrator eller inte så lägger vi bara in följande i profilen:

cd c:\scripts

Slut på Övning 31!

Det var allt för denna övning.
Har du några funderingar kring den?

Fundera på det och säg till om du vill ha hjälp!

Övning 32 – Skripta mot Active Directory

Tid: xx minuter

Den här övningen är en ren extrauppgift.

Syftet är att för de som hinner hit ska kunna omsätta det teoretiska i något praktiskt.

Denna övning MÅSTE köras i labbmiljön och fungerar att köra från klienten.
Den vi använt hittills.

Extrauppgift 1 – Skapa upp lab användare och datorer

För att vi ska kunna ha användare och datorer att jobba mot så ska vi skapa upp dom med ett skript.

1. I kursmaterialet under **03_Demon** så finns en katalog som heter AD.
2. Kopiera innehållet till **c:\scripts**.
3. Öppna upp filen **LabSetup.ps1** i Visual Studio Code.
4. Tryck på **F5**.

Nu kommer skriptet att köras en stund och det kommer att skapas många användare och många datorer.

Extrauppgift 2 – Lista alla användare

För att göra saker mot Active Directory kan man använda olika metoder.
Vi gör det enkelt för oss använder Microsofts Active Directory modul.

1. Skapa ett nytt skript.

2. Skriv
   Import-Module ActiveDirectory

Frågor:

1. Hur listar man alla kommandon som följer med ActiveDirectory modulen?
2. Vilken cmdlet ska man använda för att lista alla användare?
3. Vilken parameter ska man använda för att lista alla användare?

Extrauppgift 3 – Lista alla datorer

Frågor:

1. Utifrån föregående uppgift hur får man istället fram alla datorer?

Extrauppgift 4 – Lista alla användare från Sverige

Frågor:

1. Om du bara vill få ut användare från Sverige, hur gör du då?

Extrauppgift 5 – Lista alla användare skapade innan dagens datum

Frågor:

1. Om du vill lista användare skapade innan dagens datum hur gör du då?

Extrauppgift 6 – Skapa användare

Frågor:

1. Hur gör man om man vill skapa användare direkt i **LabOU**t?

   Ett tips använd hjälpen för **New-ADUser**.

   Och vill du ha mera hjälp så titta i LabSetup.ps1 skriptet och sök på **New-ADUser**.

   Skriptet finns mycket för att man ska kunna hitta hjälp och tankar av det!

Det var alla uppgifter.

Många extrauppgifter

MEN vill du ha ännu lite extra fundera på och gör detta:

1. Hur gör man för att spara alla användare i Danmark i ett format som man kan läsa in i excel?
2. Hur gör man för att sortera dem i bokstavsordning innan man spar dem?
3. Hur sorterar man dem när man har dem i t ex csv format?
4. Hur många användare finns det i Sverige och Danmark?
5. Hur många användare finns det totalt i Active Directory?
6. Hur listar man alla OUn i OUt LabOU?