Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Správa Windows serveru za využití PowerShell

Podpůrné materiály v podobě video tutoriálů

Bakalářská práce

Autor: Radek Dušek
Studijní obor: im3-p

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Hradec Králové Duben 2024

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

……

V Hradci Králové dne 26.4.2024 Radek Dušek

Poděkování:

Děkuji vedoucímu bakalářské práce Mgr. Josefu Horálkovi, Ph.D. za metodické vedení práce a za jeho trpělivost. Chtěl bych mu také poděkovat za poskytnutí materiálů a možnost pravidelných konzultací.

Anotace

Tato práce se zabývá tvorbou podpůrných výukových materiálů na téma správy Windows serveru za využití programu PowerShell ve formě video tutoriálů.
V první teoretické části se autor zabývá obecným představením PowerShellu a jeho teoretických možností využití k ovládání a správě Windows serveru.
V druhé praktické části autor navrhuje 7 praktických dílčích úloh, zaměřených na konkrétní oblasti správy Windows Serveru. Ke každé takové úloze autor poskytuje řešení problému ve formě okomentovaných PowerShell skriptů.
Dále jsou v rámci této praktické části práce vytvořeny video tutoriály, ve kterých je ukázán a popsán postup řešení těchto dílčích úloh. Tyto video tutoriály by měly sloužit jako podpůrné materiály pro výuku předmětu operační systémy.

Annotation

Title: Managing a Windows server using PowerShell

This thesis deals with the creation of supporting tutorials on Windows server administration using PowerShell in the form of video tutorials.
In the first theoretical part, the author deals with a general introduction of PowerShell and its theoretical possibilities of using it to control and manage Windows server.
In the second practical part, the author proposes 7 practical tasks, focused on specific areas of Windows Server administration.
For each such task, the author provides a solution to the problem in the form of commented PowerShell scripts.
Furthermore, video tutorials are created within this practical part of the work, in which the procedure of solving these subtasks is shown and described.
These video tutorials should be used as support materials for teaching the subject of operating systems.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Pro serverové administrátory je PowerShell důležitý nástroj díky své skriptovací schopnosti k automatizaci rutinních úloh, což umožňuje šetřit čas a minimalizovat chyby. Díky své úzké provázanosti s prostředím Windows je skvělým výběrem pro správu Windows Serveru. Mojí snahou je v této práci nejen tyto schopnosti představit jak v teoretické rovině, tak následně v rovině praktické.

2 Cíl práce

Smyslem této práce je představení PowerShellu a jeho možného využití v oblasti správy Windows Serveru.

Hlavním cílem je vytvoření podpůrných výukových materiálů ve formě praktických úkolů z různých oblastí správy Windows Serveru a souvisejících řešení ve formě PowerShell skriptů a video tutoriálů.

3 Metodika zpracování

Cílem bylo představit a ukázat možnosti využití PowerShellu v oblasti správy a automatizace správy Windows Serveru.
V rámci vlastního výzkumu jsem čerpal informace zejména z oficiální PowerShell dokumentace, kterou Microsoft zdarma poskytuje na svých stránkách.
Dále jsem čerpal informace z různých internetových odborných i neodborných článků a diskusních fór.
V rámci rešerše jsem se seznámil s několika odbornými pracemi na téma PowerShell, ale informace z žádné z nich jsem nakonec nevyužil, jelikož se netýkaly tématu správy Windows Serveru.
Dále jsem se v rámci rešerše seznámil s knihou 'PowerShell' (Ed Wilson,2022) ISBN:978-80-251-5049-8.

4 PowerShell

PowerShell je open-source multiplatformní program pro automatizaci úloh a správu konfigurace od společnosti Microsoft. Skládá se z příkazového řádku, skriptovacího jazyka a rámce pro správu konfigurace. Kromě operačního systému Windows, jehož je PowerShell výchozí automatickou součástí, lze PowerShell nainstalovat také na systémy Linux, macOS a další. [1]

PowerShell je založen na platformě .NET. Díky této platformě má PowerShell přístup k .NET třídám a lze je tedy využívat přímo z PowerShellu.

4.1 Historie

Vývoj PowerShellu začal už v roce 2002 (tehdy pod názvem Monad) ale první oficiální verze PowerShell 1.0 vyšla 14. listopadu 2006. [6]

Od verze 2.0 se Windows PowerShell stal automatickou předinstalovanou součástí systémů Windows a Windows Server, konkrétně Windows 7 a Windows Server 2008R2 a také všech budoucích verzí. Bylo ho ale také možné doinstalovat na starší verze Windows XP, Windows Server 2003, Windows Vista a Windows Server 2008. [6]

Od verze Windows 10 PowerShell úplně nahradil Příkazový řádek (cmd.exe) a stal se výchozím příkazovým prostředím Průzkumníka souborů. [11] [12]

Původně byl PowerShell určen pouze pro systémy Windows, ale 18. srpna 2016 společnost Microsoft oznámila, že vytvořili PowerShell jako open-source a multiplatformní s podporou pro Windows, macOS, CentOS a Ubuntu. [13] Zdrojový kód byl zveřejněn na GitHubu. [14] Tato nová verze se nazývala PowerShell Core 6.0. S přechodem na open source vznikla nová generace Windows PowerShellu, která byla přejmenována na "PowerShell Core“ a běžela nově na platformě .NET Core, namísto do té doby používaného .NET Framework. [6]

Poslední verze primárně určená pro Windows, PowerShell 5.1, vyšla v lednu 2017. Momentálně je nejnovější verze PowerShell 7.4, ale verze 5.1 zůstává i nadále funkční a v prostředí Microsoft je stále nejrozšířenější. [2]

Od verze 7.0 byl PowerShell Core přejmenován pouze na „PowerShell“ a platforma .NET Core byla přejmenována pouze na „ .NET“. [3]

4.2 Proč PowerShell

Proč použít pro správu Windows Serveru právě PowerShell?

Největší česká IT akademie a Ministerstvem školství akreditované vzdělávací zařízení ITnetwork shrnuje výhody použití PowerShellu v několika bodech:

- „Dá se velice snadno naučit a pochopit.
- Lze jej lehce kombinovat s různými nástroji a aplikacemi (API, databáze, správa operačních systémů).
- Je primárně určen k automatizaci a tuto práci odvádí skvěle.
- Možnost tvorby vlastních modulů, a tudíž rozšiřitelnost o vlastní funkcionality.
- Je naprosto zdarma, jediná investice je čas a chuť se jej naučit.
- Zájem o PowerShell raketově roste, tudíž jeho znalost při hledání práce je velké plus.
- V dnešní době je v top žebříčku skriptovacích jazyků nejen pro Windows prostředí.“ [2]

Mezi další výhody PowerShellu lze zařadit:

-Je výchozí součástí systémů Windows [6]

-Je multiplatformní a open-source [6]

-Pracuje s objekty [6]

-Je založen na .NET [6]

-Kvalitní standardizovaná dokumentace [15]

-Funkce doplňování kódu, konzolová nápověda, standardizované pojmenování rutin, historie příkazového řádku, podpora aliasů [15] [3]

-Podpora pipeline pro zřetězování příkazů [3]

-Rozšiřitelnost pomocí funkcí, profilů, skriptů a modulů [3]

-Integrovaná podpora běžných formátů, jako CSV, JSON, XML [3]

-Vzdálená správa – PowerShell umožňuje spravovat jedno či více zařízení vzdáleně [16]

Většina z těchto výhod je podrobněji popsána dále v práci.

4.3 Rutiny, parametry a argumenty

Díky provázanosti s .NET platformou poskytuje PowerShell velké množství funkcí pro správu v podobě příkazů - tzv. cmdlets (česky rutiny). Rutiny jsou specializované .NET třídy implementující určitou operaci, které PowerShell vytváří a volá za běhu. Kolekci rutin společně s logickými podmínkami označujeme jako PowerShell skript. [4][5] PowerShell skripty mají standardně příponu .ps1.

Včetně rutin lze v PowerShellu spustit všechny příkazy podporované daným systémem. Mezi další spustitelné příkazy v PowerShellu patří spustitelné soubory (např. ping.exe), ty jsou ale na rozdíl od rutin spouštěny jako samostatný oddělený proces. [1]

Rutiny jsou v PowerShellu standardně pojmenovány pomocí dvojice slov sloveso-podstatné jméno, kde sloveso identifikuje akci, kterou rutina provede, následuje znak pomlčka a podstatné jméno, které identifikuje prostředek, na kterém rutina provádí svou akci. Použité sloveso by mělo být jedno ze seznamu povolených sloves, tento seznam lze zobrazit pomocí rutiny *Get-Verb* nebo jej lze nalézt v Microsoft dokumentaci. [7] Použité podstatné jméno by mělo být vždy ve formě jednotného čísla. [2] Například rutina *Get-Command* vypíše seznam všech příkazů podporovaných daným systémem. Tato konvence pojmenování rutin usnadňuje pochopení, co daná rutina dělá a také to usnadňuje vyhledávání potřebných rutin. [3]

Díky tomu je PowerShell vhodný i pro úplně začátečníky, jelikož k práci s ním není potřeba znát jiné skriptovací či programovací jazyky.

Výstupem rutin jsou .NET objekty (případně kolekce objektů). Objekt .NET dokáže rutina přijmout i jako vstupní hodnotu, což umožňuje zřetězení. [3]

Rutiny se jako třídy dědí z .NET třídy *Cmdlet*, případně z třídy *PSCmdlet* pokud rutina potřebuje interagovat s běhovým prostředím PowerShellu. Tyto základní třídy specifikují určité metody: *BeginProcessing()*, *ProcessRecord()*, *EndProcessing()*, které rutiny dědí a přetěžují dle své konkrétní funkce. Kdykoliv se spustí rutina, PowerShell zavolá tyto metody v pořadí po sobě, tak jak jsou uvedeny (metodu *ProcessRecord()* zavolá pouze v případě, když obdrží vstup

pipeliny). Dále musí mít třída implementující rutinu .NET atribut *CmdletAttribute*, který specifikuje sloveso a podstatné jméno tvořící název rutiny. [6]
Rutiny mohou být napsány v libovolném zkompilovaném .NET jazyce (například C#, PHP, a další) nebo přímo ve skriptovacím jazyce PowerShellu. Microsoft poskytuje dokumentaci pro vývojáře vlastních rutin, funkcí a modulů, včetně pravidel, kterých by se při vývoji měli držet. [8]

Parametry rutiny jsou nástrojem umožňujícím rutině přijímat vstupní informace. **Argumenty** (též nazývané hodnoty) těchto parametrů specifikují vstup pro rutinu, definují způsob vykonání jejích akcí a určují data, která jsou následně vrácena do roury. [3]
V PowerShellu rozlišujeme pět druhů parametrů:

1) **Pojmenované parametry:** Vyžadují zadání názvu parametru a odpovídajícího argumentu při volání rutiny. Ve výchozím nastavení jsou všechny parametry rutiny pojmenované.
2) **Poziční parametry:** Vyžadují zadání argumentů v relativním pořadí. Systém namapuje argumenty na odpovídající poziční parametry.
3) **Povinné a nepovinné parametry:** Povinné parametry musí být zadány před voláním rutiny. Pokud není povinný parametr specifikován, PowerShell vyvolá chybu. Nepovinné parametry jsou volitelné a nemusejí být zadány při volání rutiny. Ve výchozím nastavení jsou parametry definovány jako nepovinné.
4) **Přepínací parametry:** Automaticky se nastaví na false, pokud nejsou při volání rutiny zadány.

Při správném návrhu rutiny se doporučuje deklarovat nejpoužívanější parametry jako poziční. To umožní uživateli spouštět rutinu bez nutnosti zadávat názvy parametrů. [3]
Poziční i pojmenované parametry mohou přijímat jednotlivé argumenty nebo více argumentů oddělených čárkami. Možnost zadání více argumentů je povolena pouze v případě, že parametr přijímá kolekci, například pole řetězců. V jedné rutině lze kombinovat poziční a pojmenované parametry, kdy systém nejprve

zpracuje pojmenované argumenty a poté se snaží namapovat zbývající nepojmenované argumenty na poziční parametry. [3]

Příklad:

```
Get-Process -Name "explorer"
```

Obr. 1 Příklad příkazu pro získání informací o procesu „explorer“.

Zdroj: vlastní zpracování

Na obrázku lze vidět příkaz, který zobrazí informace o procesu explorer.

Get-Process je rutina, slouží k získání informací o všech běžících procesech.

-Name je parametr, upřesňuje předchozí rutinu k zobrazení informací pouze o procesu s konkrétním názvem.

"explorer" je argument, je to název konkrétního procesu o němž se mají zobrazit informace.

4.4 Objektovost a pipeline

Jednou z výhod PowerShellu oproti ostatním populárním shellům jako jsou např. bash nebo Windows Command Line je, že PowerShell je **objektový**, dokáže tedy pracovat s objekty. Všechny objekty v PowerShellu jsou objekty .NET, díky tomu všechny obsahují metodu *ToString()*, kterou PowerShell používá pro reprezentaci objektů v textové podobě. Objekty lze v PowerShellu zasílat pomocí rour včetně všech jejich vlastností a metod.

Pipeline (česky roura) je jedna z nejužitečnějších funkcionalit celého PowerShellu. Pipeline v PowerShellu umožňuje zapojení více příkazů ke zpracování jednoho zdroje dat tak, že předchozí příkaz předává svůj výstup dalšímu příkazu ke zpracování a takto se to opakuje až do zobrazení výsledku. Pro použití pipeline musí být jednotlivé příkazy odděleny znakem svislé čáry „|“ (někdy též nazýváno „pipeline“ či „pipe“).

```
Get-Service | Select-Object -Property DisplayName
```

Obr. 2 Ukázka použití roury na jednoduchém příkazu.

Zdroj: vlastní zpracování

Na ukázce lze vidět jednoduché použití roury. Příkaz v ukázce vybere seznam služeb běžících na počítači a poté zobrazí pouze jejich názvy. Rutina *Get-Service* slouží k získání všech služeb na počítači, pomocí pipeline „|“ je poté výsledek předán rutině *Select-Object*, která díky parametru *-Property DisplayName* zobrazí pouze názvy těchto služeb.

Pipeline v PowerShellu je objektová, což znamená, že data procházejí pipelinou ve formě objektů na místo jednotlivých bytů, jako je tomu například v UNIXových shellech, kde je pipeline textová. Dále se pipeline v PowerShellu od té unixové liší tím, že se v PowerShellu jednotlivé fáze spouštějí v rámci běhového prostředí, a nikoli jako sada procesů koordinovaných operačním systémem. Použití objektů a provádění fází v rámci běhového prostředí PowerShell eliminuje potřebu serializovat datové struktury nebo je získávat explicitně rozborem textového výstupu. Objekt může také zapouzdřit určité funkce, které pracují s obsaženými daty a které se zpřístupní příjemci k použití. [6] V případě, že se v pipelině objeví kolekce více objektů, PowerShell postupně zavolá následující rutinu na každý objekt zvlášť. [1] Při poslední rutině v pipelině PowerShell automaticky předává její výstupní objekt rutině *Out-Default*, která objekty transformuje do streamu formátovaných objektů a ty pak vykreslí na obrazovku. [6]

4.5 Moduly

PowerShell je modulový, což dává možnost pomocí přídavných modulů PowerShell rozšířit o další funkcionality. Modul v PowerShellu představuje soubor vlastních funkcí, proměnných, tříd nebo rutin. Takový soubor lze do aktuální PowerShell relace importovat příkazem *Import-Module* a využívat tak všechny jeho funkcionality v této relaci. [2]

V PowerShellu rozlišujeme několik druhů modulů:

1) **Script moduly**: Soubory obsahující nezkompilovaný kód. Díky tomu jsou velmi rozšířené a oblíbené a jsou přístupné všem. Tyto moduly mají příponu .psm1.
2) **Binární moduly**: Tyto moduly obsahují zkompilovaný kód v jazyce C#. Mají příponu .dll.

3) **Manifest moduly**: Nejsou přímo moduly, ale slouží jako datový soubor pro script moduly. Při importu PowerShell modulů plní manifest soubor klíčovou roli jako řídící prvek. Definuje například, co má být exportováno z modulu, stanovuje minimální verzi PowerShellu nutnou pro jeho import a další parametry. Struktura manifestu je pevně stanovena a kompletní dokumentaci lze nalézt na oficiálních stránkách Microsoftu. [10] Mají příponu .psd1.
4) **Dynamické moduly**: Tyto moduly existují pouze v paměti a jsou specifické pro konkrétní relaci PowerShellu.

PowerShell má v sobě vestavěnou funkcionalitu pro automatické načítání modulů Module Autoload. Tato funkce automaticky importuje modul, pokud se nachází v požadované cestě. Cesty, které PowerShell sleduje pro module autoload, jsou určeny v proměnné $env:PSModulePath. Pokud je modul umístěn v jedné z těchto lokalit, můžeme jej importovat pouhým zavoláním jeho jména. V případě, že se nachází mimo tyto složky, musíme provést import pomocí cesty. Module autoload se aktivuje v okamžiku, kdy z daného modulu zavoláme libovolnou funkci nebo rutinu. [2]

PowerShell moduly lze dohledat z různých zdrojů, ale přímo Microsoft nabízí oficiální zdroj PowerShell Gallery. [9] Z této oficiální platformy je možné nejen stahovat z více než 12 tisíc modulů (to lze buď přímo z webového klienta nebo přímo z PowerShellu), ale také se podílet na vývoji nových modulů. [2]

4.6 Aliasy

Alias je alternativní označení nebo přezdívka rutiny nebo prvku příkazu, například funkce, skriptu, souboru nebo spustitelného souboru. V prostředí PowerShellu lze alias použít místo standardního názvu příkazu. Vytvoření aliasu probíhá pomocí rutiny *New-Alias*. [3]

```
New-Alias -Name "SeznamProcesu" -Value "Get-Process"
```

Obr. 3 Ukázka vytvoření aliasu.

Zdroj: vlastní zpracování

Příkaz v ukázce vytvoří alias s názvem "SeznamProcesu", který bude odkazovat na příkaz Get-Process. Při zadání SeznamProcesu, PowerShell bude rozumět, že je to ekvivalent rutině Get-Process.

Prostředí PowerShell obsahuje sadu předdefinovaných aliasů, mezi nejznámější patří například „cd“, který zastupuje rutinu Set-Location, nebo „dir“, který systémech Windows ale i Linux a macOS zastupuje rutinu Get-ChildItem. Pro získání seznamu všech aliasů na daném počítači, včetně těch vestavěných, stačí zadat rutinu *Get-Alias*. [3]

Vytvořené aliasy jsou platné pouze v rámci aktuální relace. I z tohoto důvodu přímo Microsoft nedoporučuje používat aliasy v PowerShell skriptech. Pro jejich dostupnost i v jiných relacích je potřeba přidat aliasy do svého profilu PowerShellu. Alternativně lze použít rutinu *Export-Alias* k uložení aliasů do souboru. [3]

Alias lze přiřadit rutině, skriptu, funkci nebo spustitelnému souboru. Alias lze v příkazech používat stejně jako plný název rutiny a lze jej používat i s parametry. Alias ale nelze přiřadit přímo rutině a jejím parametrům (alias tedy nemůže zastupovat rutinu včetně parametrů). Pro dosáhnutí podobné funkcionality je třeba vytvořit funkci, která obsahuje rutinu s parametry. [3]

4.7 Funkce

Funkce v PowerShellu je seznam příkazů s přiřazeným názvem. Pro spuštění funkce stačí zadat její jméno, a příkazy v seznamu se automaticky provedou.

Podobně jako rutiny, i funkce mohou obsahovat parametry, které mohou být pojmenované, poziční, přepínací nebo dynamické. [3]

Funkce mohou vracet hodnoty, které lze zobrazit, přiřadit proměnným nebo předat jiným funkcím nebo rutinám. Návratovou hodnotu lze také specifikovat pomocí klíčového slova *return*. Klíčové slovo return neovlivňuje ani nepotlačuje jiný výstup generovaný funkcí, pouze ukončuje vykonávání funkce na daném řádku. [3]

Funkce mohou fungovat podobně jako rutiny. To umožňuje vytvořit funkci, která funguje stejně jako rutina, bez nutnosti znalosti programování v žádném .NET jazyce. [3]

Funkci lze vytvořit pomocí klíčového slova *function.*

```
function <název_funkce> {příkazy}
```

Obr. 4 Ukázka syntaxe vytvoření jednoduché funkce.

Zdroj: vlastní zpracování

Mimo názvu funkce je při založení potřeba zadat jednotlivé příkazy funkce. V případě více než jednoho příkazu je potřeba jednotlivé příkazy psát každý na samostatný řádek, případně je oddělovat středníkem. [3]

Vytvoření složitější funkce, která dokáže přijmout parametry a pracovat v pipelině může vypadat následovně:

```
function <název_funkce>
{
  param([typ]$parametr1 [,[typ]$parametr2])
  begin {<seznam příkazů>}
  process {<seznam příkazů>}
  end {<seznam příkazů>}
  clean {<seznam příkazů>}
}
```

Obr.5 Ukázka syntaxe vytvoření komplexní funkce.

Zdroj: vlastní zpracování

Za klíčové slovo *param* lze dosadit libovolné množství parametrů, které funkce bude schopna přijmout. Bloky kódu *begin*, *process*, *end*, *clean* slouží ke zpracování vstupu z pipeliny. Blok *begin* se spustí pouze jednou na začátku funkce, blok *process* se spustí jednou pro každý objekt v pipelině. Poté, co funkce obdrží všechny objekty v pipelině, proběhne jednou blok *end*. Ve verzi PowerShell 7.3 přibyl blok *clean*, ten poskytuje uživateli pohodlný způsob pro vyčištění prostředků použitých v předchozích blocích. [3]

Použití těchto bloků je volitelné, nemusí být použity všechny. V případě ale, že je některý z těchto bloků definován, musí být všechny příkazy uvnitř těchto bloků. Jakýkoli kód mimo tyto bloky nebude rozpoznán. Pokud funkce nepoužívá žádný z těchto bloků, je se všemi příkazy zacházeno jako by se nacházely v koncovém bloku *end*. [3]

Praktickou ukázku na vytvoření a použití funkce v PowerShellu lze ukázat na následujícím příkladu:

```
function Obvod_Kruhu
{
    param([double]$Poloměr)

    $Pí = [Math]::PI
    $Obvod = 2 * $Pí * $Poloměr

    Write-Output "Obvod kruhu je: $Obvod"
}
```

Obr.6 Ukázka vytvoření funkce pro výpočet obvodu kruhu.

Zdroj: vlastní zpracování

V této ukázce lze vidět syntaxi vytvoření funkce pro výpočet obvodu kruhu. Jako vstupní parametr je zde použit poloměr kruhu, který je datového typu double. Tento datový typ umožňuje použití čísel s desetinnými místy. Dále je proměnné Pí přiřazena hodnota čísla pí. Ta je získána ze třídy Math a její statické metody PI. Poté je vzorečkem vypočítán obvod a vypsán do konzole.

Pro zavolání této funkce stačí zadat její název včetně parametru.

```
Obvod_Kruhu -Poloměr 3.74

Obvod kruhu je: 23.4991130488517
```

Obr.7 Ukázka zavolání funkce a jejího výstupu.

Zdroj: vlastní zpracování

Při definici funkce v PowerShellu je tato funkce dostupná pouze v rámci aktuální relace. Funkce zůstane dostupná až do ukončení této relace. Pro zajištění dostupnosti funkce napříč všemi relacemi prostředí PowerShell je třeba ji přidat do svého PowerShell profilu. Alternativně lze funkci uložit do PowerShell skriptového souboru pro snadnou opakovatelnost a sdílení. Všechny funkce a filtry v prostředí PowerShell jsou automaticky uloženy v jednotce Function:. Pro zobrazení všech funkcí v aktuální relaci stačí zadat příkaz Get-ChildItem function:. [3]

4.8 Profily

Mnoho prvků vytvořených nebo spuštěných v PowerShellu ovlivňuje pouze aktuální relaci. Po ukončení relace jsou tyto prvky odstraněny. Mezi prvky specifické pro relaci patří proměnné, aliasy, funkce, příkazy a moduly PowerShellu, přidané do aktuální relace. Pro uchování těchto prvků a jejich dostupnost ve všech budoucích relacích je potřeba je přidat do svého PowerShell profilu. [3]

Profil v prostředí PowerShellu představuje skript, který se spouští při spuštění PowerShellu. Tento profil lze využít jako spouštěcí skript k úpravě chování prostředí dle svých potřeb. Lze zde přidávat příkazy, aliasy, funkce, proměnné, moduly, PowerShell jednotky a další. Dále je možné do profilu zařadit další relačně specifické prvky, čímž budou dostupné v každé nové relaci bez nutnosti opakovaného importu nebo opětovného vytváření. [3]

PowerShell podporuje 4 kategorie profilů dle působnosti na uživatele a hostitelské programy. Hostitelskými programy je zde myšleno nejčastěji používaný console host (terminál) a PowerShell ISE (Integrated Scripting Enviroment). PowerShell lze také spustit v editoru Visual Studio Code od Microsoftu. [2]

Zároveň PowerShell obsahuje proměnnou *$PROFILE*, která automaticky uchovává cesty profilům, které jsou k dispozici v aktuální relaci.

1) **Všichni uživatelé, všichni hosté** (All users, all hosts)
   Profil, který ovlivňuje všechny hostitelské programy, bez ohledu na přihlášeného uživatele. Cestu lze získat pomocí proměnné *$PROFILE.AllUsersAllHosts*
2) **Všichni uživatelé, konkrétní host** (All users, current host)
   Profil, který ovlivňuje konkrétní hostitelský program nezávisle na přihlášeném uživateli. Cestu lze získat pomocí proměnné *$PROFILE.AllUsersCurrentHost*
3) **Konkrétní uživatel, všichni hosté** (Current user, all hosts)
   Tento profil ovlivňuje všechny hostitelské programy a konkrétního uživatele. Cesta uložena v proměnné *$PROFILE.CurrentUserAllHosts*.
4) **Konkrétní uživatel, konkrétní host** (Current user, current host)
   Jedná se o nejpoužívanější kategorii profilu, jelikož je aplikován pouze na konkrétního hosta pro konkrétního uživatele. Jeho cesta je uložena v proměnné $PROFILE (případně $PROFILE.CurrentUserCurrentHost) [2]

Skripty profilů jsou spouštěny v uvedeném pořadí, což znamená, že úpravy provedené v profilu AllUsersAllHosts mohou být přepsány jakýmkoli jiným následujícím skriptem profilu. Profil CurrentUserCurrentHost je vždy spouštěn jako poslední. [3]

Nový profil lze vytvořit příkazem *New-Item $PROFILE*, kde $PROFILE specifikuje konkrétní typ profilu. Tímto se vytvoří prázdný skript, který je potřeba nakonfigurovat. Ten lze jednoduše konfigurovat v libovolném textovém editoru, nebo přímo v PowerShellu. [2] [3]

Praktická ukázka vytvoření profilu může vypadat následovně:

```
New-Item $PROFILE
```

Obr.8 Ukázka vytvoření profilu.

Zdroj: vlastní zpracování

Tento příkaz vytvoří profil pro konkrétního uživatele a konkrétního hosta. Tento profil je následně třeba nakonfigurovat. To lze například v textovém editoru notepad. Otevřít profil v notepadu lze následujícím příkazem.

```
notepad $PROFILE
```

Obr.9 Ukázka otevření profilu v notepadu.

Zdroj: vlastní zpracování

Poté stačí do profilu přidat vše co chceme opakovaně používat a soubor uložit. Po zavření a opětovném otevření Powershellu budou všechny provedené úpravy profilu aktivní a dostupné pro použití.

4.8.1 Zásady spouštění

Zásady spouštění (Execution Policy) v PowerShellu řídí, zda je umožněno spouštění skriptů a načítání konfiguračních souborů, včetně profilů. Ve výchozí konfiguraci je nastavena zásada omezeného spouštění, která zabrání spuštění všech skriptů, včetně profilů. Ponecháním zásady na „Omezené“ se profil nespustí a jeho obsah nebude aplikován. [3]

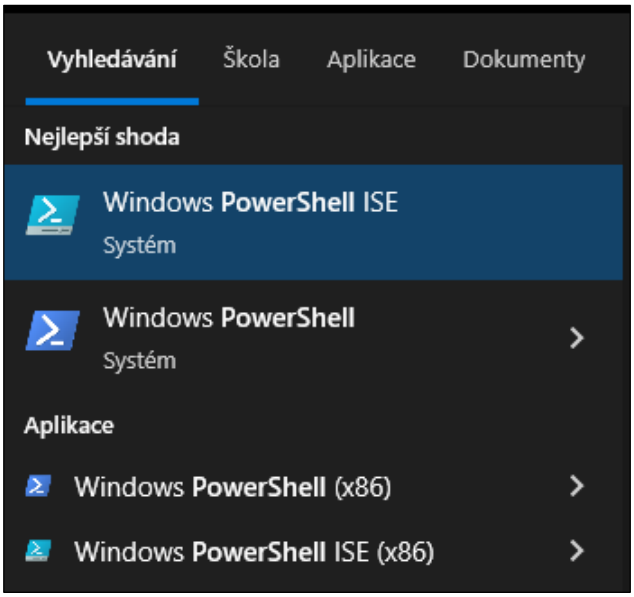
Příkaz *Set-ExecutionPolicy* slouží k nastavení a změně zásad spouštění. Jedná se o jeden z mála příkazů, který platí ve všech relacích PowerShellu, protože hodnota je uložena v registru. Není nutné ji nastavovat při otevírání PowerShellu a příkaz Set-ExecutionPolicy není potřeba ukládat do svého profilu.

Pro změnu zásad spouštění, tak aby mohly být spouštěny profily je třeba zadat příkaz Set-ExecutionPolicy AllSigned. [2] [3]

4.9 Spuštění Powershellu

Jelikož je tato práce zaměřená na systémy Windows a Windows Server, není nutné Powershell nijak instalovat, ten je výchozí součástí těchto systémů.

Pro spuštění stačí do vyhledávacího panelu zadat termín „powershell“ a zvolit preferovanou variantu.

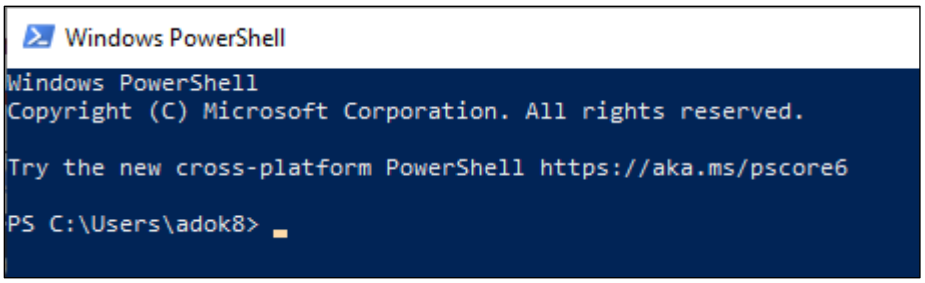


Obr.10 Ukázka vyhledání powershellu ve Windows 10.

Zdroj: vlastní zpracování

Na výběr je ze dvou variant, a to Powershell a Powershell ISE. Dále jsou zde tyto dvě varianty i v 32bitové verzi označené příponou (x86).

Verze s označením „Windows PowerShell“ je základní Powershell konzole, též označována jako „terminál“. Tato verze je vhodná spíše pro jednodušší práci jako volání skriptů a jednořádkové příkazy. [2]

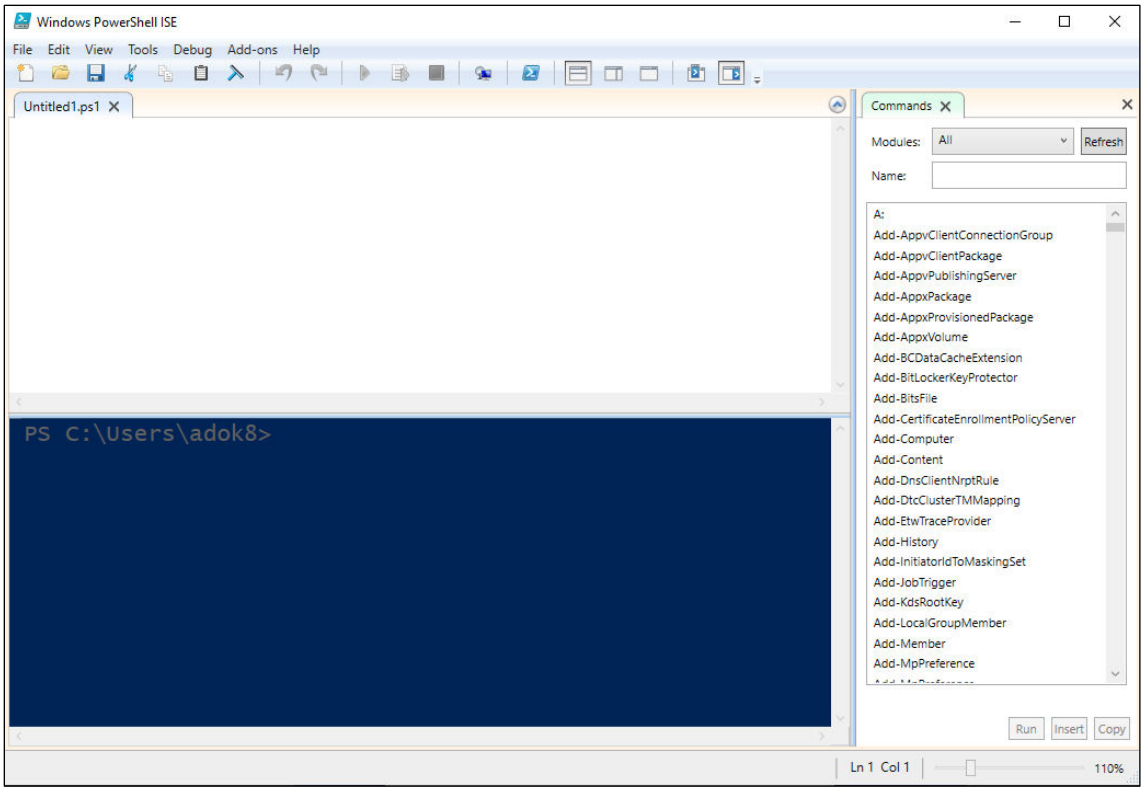


Obr.11 Ukázka Powershell konzole.

Zdroj: vlastní zpracování

Powershell ISE (Integrated Scripting Enviroment) je integrované skriptovací prostředí s grafickým uživatelským rozhraním pro Powershell. Toto prostředí je vhodné i pro složitější skriptování v Powershellu, jelikož umožňuje spouštět příkazy a psát, testovat a debugovat skripty. Dále poskytuje funkcionality jako jsou víceřádkové úpravy, automatické doplňování příkazů, barvení syntaxe, selektivní

spouštění části kódu, kontextovou nápovědu a podporu překladu do různých světových jazyků. [3]



Obr.12 Ukázka prostředí Powershell ISE.

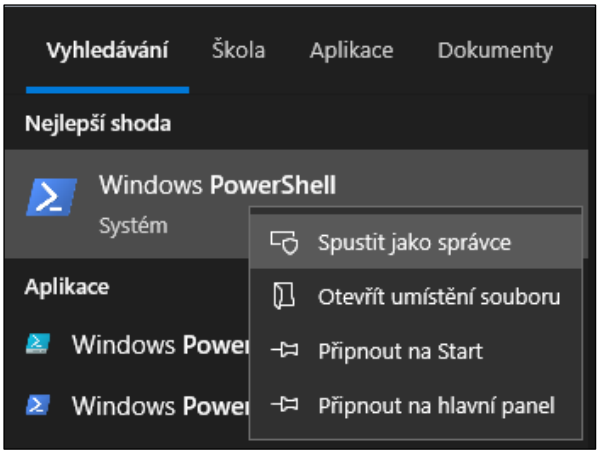
Zdroj: vlastní zpracování

Na vrchní části obrazovky se nachází panel s nástroji pro přizpůsobení vzhledu samotného prostředí, ale také některé funkcionality. Pod ním se nachází samotný textový editor – hlavní skriptovací část. V dolní polovině obrazovky se nachází samotná konzole. V pravé části lze zobrazit explorer příkazů, který umožňuje prozkoumat všechny příkazy včetně nápovědy k nim. Tento explorer ale lze v prostředí skrýt a není nutné ho používat.

Automatické doplňování příkazů se provádí klávesou Tab. Pravým kliknutím v kódu lze nastavit breakpoint, u kterého se zastaví provádění kódu. Označením části kódu myší lze spustit pouze tuto vybranou část. Kontextová nápověda pro příkaz lze zobrazit klávesou F1.

Při výše popsaném postupu spuštění se PowerShell spustí s běžnými uživatelskými oprávněními. To neumožňuje provádět některé akce, které vyžadují oprávnění správce, jako je například přístup k systémovým složkám a souborům.

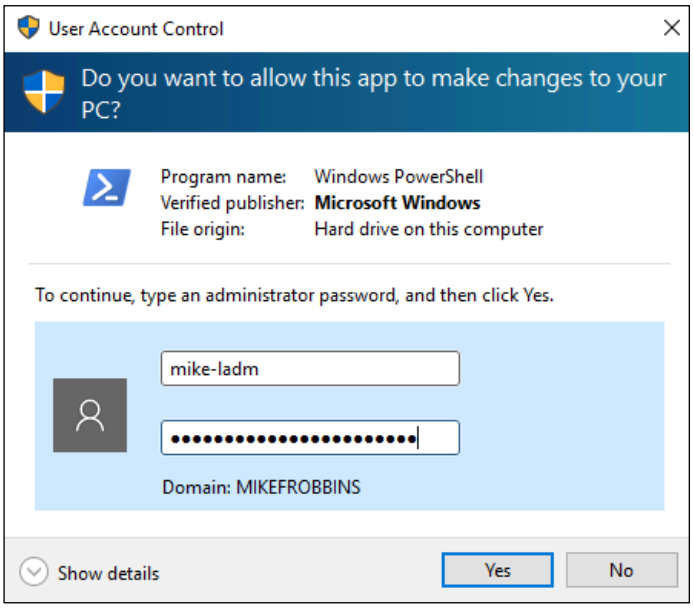
Ke spuštění PowerShellu s vyšší úrovní oprávnění je třeba ho spustit „jako správce“. Toho lze docílit pravým kliknutím a výběrem příslušné možnosti.



Obr.13 Ukázka spuštění PowerShellu s vyšší úrovní oprávnění.

Zdroj: vlastní zpracování

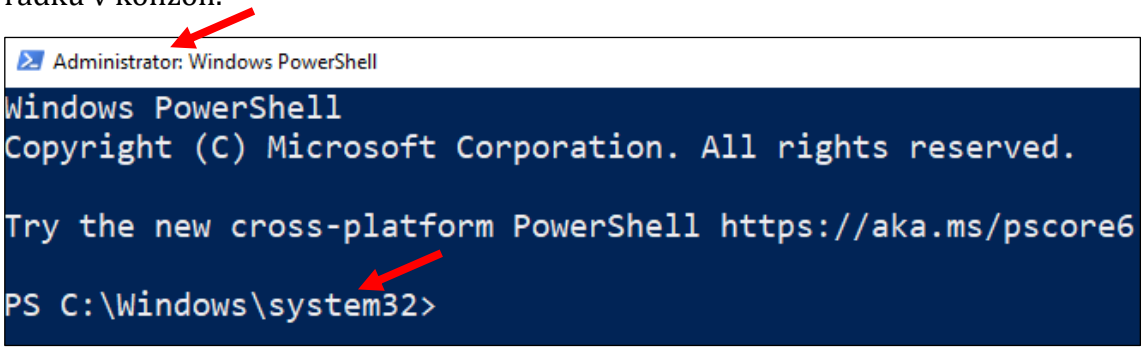
Uživatel, který není ve Windows přihlášen jako správce bude následně vyzván k zadání přihlašovacích údajů správce.



Obr.14 Ukázka dialogového okna výzvy ve Windows.

Zdroj: https://learn.microsoft.com/en-us/powershell/scripting/learn/ps101/01-getting-started?view=powershell-7.4

Že je PowerShell spuštěn jako správce lze poznat podle záhlaví aplikace, nebo dle řádku v konzoli.



Obr.15 Ukázka PowerShellu spuštěného jako správce.

Zdroj: Vlastní zpracování

Stejně to platí i pro prostředí ISE.

Při práci v PowerShellu s takto zvýšenou úrovní oprávnění je třeba být opatrný. Veškeré příkazy jsou spouštěny s administrátorskými oprávněními.

4.10 Skriptování

PowerShell nabízí dynamicky typovaný skriptovací jazyk, který umožňuje imperativní implementaci složitých operací pomocí rutin. Tento skriptovací jazyk podporuje proměnné, funkce, větvení, smyčky, strukturované zpracování chyb a výjimek, uzávěry a lambda výrazy, a také integraci s .NET rámcem. Proměnné ve skriptech PowerShellu mají předponu $ a nemají typovou kontrolu. Proměnným může být přiřazena libovolná hodnota, včetně výstupu rutin. Řetězce mohou být uzavřeny buď do jednoduchých ‘ nebo dvojitých “ uvozovek. Při použití dvojitých uvozovek dojde k expanzi proměnných, i když jsou uvnitř uvozovek. Uzavřením cesty k souboru do závorek, kterým předchází znak dolaru (v podobě ${C:\soubor.txt}) se vytvoří odkaz na obsah souboru. Tato notace umožňuje jak zápis do souboru, tak čtení obsahu souboru. V případě přiřazení objektu je tento objekt před uložením serializován. [6]

K metodám objektů lze přistupovat pomocí tečkové notace, stejně jako například v syntaxi jazyka C#. PowerShell poskytuje speciální proměnné jako například *$args*, což je pole všech argumentů předávaných funkci z příkazového řádku. Proměnná *$_* odkazuje na aktuální objekt v pipelině. PowerShell umí pracovat

s poli a asociativními poli. PowerShell umí také vyhodnocovat aritmetické výrazy a analyzovat běžné zkratky jako například GB, MB a KB. [6] [3]

PowerShell umožňuje zavolání libovolné statické .NET metody. K tomu stačí zadat jmenný prostor v hranatých závorkách [] a následným použitím dvojice dvojteček :: pro označení konkrétní statické metody, například: [Console]::WriteLine("PowerShell").[6]

Pro zpracování chyb poskytuje PowerShell mechanismus zpracování výjimek založený na .NET. V případě chyby je vyhozen objekt obsahující informace o chybě, které jsou zachyceny pomocí konstrukce try ... catch. Prostředí PowerShell lze nakonfigurovat tak, aby pokračovalo v provádění, aniž by skutečně došlo k vyhození výjimky. [6]

Na rozdíl od starého příkazového řádku má PowerShell možnost přistupovat nejen k souborovému systému, ale také například k registrům systému, úložišti certifikátů a dalším. [1]

PowerShell rovněž poskytuje hostitelské API rozhraní, které umožňuje běhové prostředí PowerShell vložit do jiných aplikací. Tyto aplikace pak mohou využívat PowerShell k implementaci určitých operací, včetně těch, které jsou vystaveny prostřednictvím grafického rozhraní.

Mezi aplikace, které je možno plnohodnotně spravovat přes příkazovou řádku PowerShellu se řadí například Microsoft SQL Server 2008, Microsoft Exchange Server 2007, IIS, SharePoint a další. [1]

PowerShell není case sensitive, nerozlišuje tedy velká a malá písmena.

PowerShell příkazová řádka umožňuje doplňování názvů příkazů, souborů, aliasů a dalších pomocí klávesy Tab.

PowerShell poskytuje vlastní rozsáhlou konzolovou nápovědu dostupnou pomocí rutiny *Get-Help*. Aktualizovaný obsah nápovědy lze získat z internetu pomocí rutiny *Update-Help*.

```
1  Get-Help Get-Date

NAME
    Get-Date

SYNOPSIS
    Gets the current date and time.


SYNTAX
    Get-Date [[-Date] <System.DateTime>] [-Day <System.Int32>] [-DisplayHint {Date
    lisecond <System.Int32>] [-Minute <System.Int32>] [-Month <System.Int32>] [-Sec

    Get-Date [[-Date] <System.DateTime>] [-Day <System.Int32>] [-DisplayHint {Date
    -Minute <System.Int32>] [-Month <System.Int32>] [-Second <System.Int32>] [-UFor


DESCRIPTION
    The `Get-Date` cmdlet gets a DateTime object that represents the current date o
    ral .NET and UNIX formats. You can use `Get-Date` to generate a date or time ch

    `Get-Date` uses the computer's culture settings to determine how the output is
    rmat`.


RELATED LINKS
    Online Version: https://learn.microsoft.com/powershell/module/microsoft.powersh
    ForEach-Object
    Get-Culture
    Get-Member
    New-Item
    New-TimeSpan
    Set-Date

REMARKS
    To see the examples, type: "get-help Get-Date -examples".
    For more information, type: "get-help Get-Date -detailed".
    For technical information, type: "get-help Get-Date -full".
    For online help, type: "get-help Get-Date -online"
```

Obr.16 Ukázka výstupu rutiny Get-Help.

Zdroj: vlastní zpracování

Na obrázku lze vidět ukázku PowerShell nápovědy, zde konkrétně k rutině *Get-Date*. V první části NAME je specifikován název rutiny, pro který je zobrazena nápověda. V části SYNOPSIS se zobrazuje krátký popis dané rutiny. Pod částí SYNTAX lze najít konkrétní syntaxi dané rutiny, včetně všech možných argumentů. U každého argumentu je také zobrazen požadovaný datový typ. [2] V DESCRIPTION je detailní popis dané rutiny. Pod částí RELATED LINKS lze najít další odkazy a příkazy související s původní rutinou. Tag Online Version odkazuje na online verzi nápovědy k této rutině. V poslední části REMARKS PowerShell zobrazuje další nápovědy a ukázky, jak pracovat s nápovědou pro danou rutinu.

4.10.1 Řízení toku

Stejně jako mnoho dalších jazyků PowerShell obsahuje příkazy pro podmíněné a cyklické spouštění kódu.

4.10.1.1 Podmínky

Skripty musí často činit rozhodnutí a na jejich základě provádět různé logické operace. Přesně k tomu v PowerShellu slouží příkazy if a switch.

1) **if**

```
$podmínka = $true

if ($podmínka)
{
    Write-Output "Podmínka je logické hodnoty true"
}
```

Obr.17 Ukázka základní syntaxe příkazu if.

Zdroj: vlastní zpracování

Na obrázku lze vidět příklad použití příkazu if. Příkaz if nejprve vyhodnotí výraz v závorkách. Pokud se vyhodnotí jako logicky pravdivý, tak se spustí blok kódu ve složených závorkách. Pokud by byla hodnota logicky nepravdivá, pak se tento blok kódu přeskočí.

V příkladu na obrázku je proměnné $podmínka přiřazena logická hodnota true. Podmínka if by tedy byla vyhodnocena jako logicky pravdivá a spustil by se blok kódu ve složených závorkách, zde konkrétně výpis do konzole.

Příkaz if umožňuje zadat akci nejen v případě, že je podmínka vyhodnocena jako logicky pravdivá, ale i v případě že je logicky nepravdivá. K tomu slouží else podmínka.

```powershell
$podmínka = $true

if ($podmínka)
{
    Write-Output "Podmínka je logické hodnoty true"
}
else
{
    Write-Output "Podmínka je logické hodnoty false"
}
```

Obr.18 Ukázka základní syntaxe příkazu if s podmínkou else.

Zdroj: vlastní zpracování

Příkaz if není limitován pouze na jednu podmíněnou kontrolu, podmínky else lze zřetězovat za použití elseif.

V tomto příkladu lze vidět použití příkazu if s podmínkou else a elseif. Provádění zde probíhá shora dolů. Nejprve se vyhodnotí příkaz if, pokud je vyhodnocen jako logicky nepravdivý pokračuje se k dalšímu elseif případně else v řadě.

```powershell
$podmínka = $true

if ($podmínka)
{
    Write-Output "Podmínka je logické hodnoty true"
}
elseif ($podmínka -eq $false)
{
    Write-Output "Podmínka je logické hodnoty false"
}
else
{
    Write-Output "podmínka není definována"
}
```

Obr.19 Ukázka syntaxe příkazu if s podmínkou else a elseif.

Zdroj: vlastní zpracování

Poslední else je výchozí akcí, která se provede, pokud žádná z ostatních nevrátí pravdivou hodnotu. Else je vždy poslední částí příkazu if, pokud je použito.

Příkazy if do sebe lze i vzájemně vnořovat, stačí do bloku kódu zadat další if příkaz. Tím lze dosáhnout mnohem složitější logiky.

2) **switch**

Při používání příkazu if se lze často dostat do situace, kdy je potřeba rozhodnout pro jednu z mnoha podmínek, jako například v situaci na obrázku:

```
$den = "Čtvrtek"

if ($den -eq "Pondělí") {Write-Output "Dnes je pondělí!"}
elseif ($den -eq "Úterý") {Write-Output "Dnes je úterý!"}
elseif ($den -eq "Středa") {Write-Output "Dnes je středa!"}
elseif ($den -eq "Čtvrtek") {Write-Output "Dnes je čtvrtek!"}
elseif ($den -eq "Pátek") {Write-Output "Dnes je pátek!"}
elseif ($den -eq "Sobota") {Write-Output "Dnes je sobota!"}
elseif ($den -eq "Neděle") {Write-Output "Dnes je neděle!"}
else {Write-Output "Nebyl zadán den!"}
```

Obr.20 Ukázka příkazu if s mnoha podmínkami.

Zdroj: vlastní zpracování

V situaci, kdy je potřeba porovnat více podmínek je lepší použít příkaz switch. Příkaz switch je ekvivalentní řadě if příkazů, ale je jednodušší. V příkazu switch stačí zadat výraz a ten je porovnán s několika různými hodnotami. Pokud je nalezena shoda, provede se odpovídající blok kódu.

```
$den = "Čtvrtek"

switch ($den)
{
    "Pondělí" {Write-Output "Dnes je pondělí!"}
    "Úterý" {Write-Output "Dnes je úterý!"}
    "Středa" {Write-Output "Dnes je středa!"}
    "Čtvrtek" {Write-Output "Dnes je čtvrtek!"}
    "Pátek" {Write-Output "Dnes je pátek!"}
    "Sobota" {Write-Output "Dnes je sobota!"}
    "Neděle" {Write-Output "Dnes je neděle!"}
    default {Write-Output "Není zadán den!"}
}
```

Obr.21 Ukázka použití příkazu switch.

Zdroj: vlastní zpracování

Klíčové slovo default slouží k identifikaci akce, která se má stát, pokud hodnota neodpovídá žádné z podmínek. Je to ekvivalentem else v příkazu if.

4.10.1.2 Cykly

For

Příkaz for slouží k cyklickému spouštění příkazů na základě podmíněného testu. Smyčka for spouští příkazy v bloku příkazů, zatímco je zadaná podmínka pravdivá.

```
for (<Inicializace>; <Podmínka>; <Opakování>)
{
    <Blok příkazů>
}
```

Obr.22 Základní syntaxe příkazu for.

Zdroj: vlastní zpracování

Blok Inicializace představuje inicializační část příkazu, která slouží k inicializaci proměnné s počáteční hodnotou. Jedná se o jeden nebo více příkazů, které se spouští před zahájením smyčky. Tato hodnota se následně stane základem podmínky, která se testuje v následující části příkazu.

Blok Podmínka představuje část příkazu, ve které se vyhodnocuje logická podmínka. Podmínka je vyhodnocena pří každém spuštění smyčky. Pokud je podmínka vyhodnocena jako logicky pravdivá, spustí se příkazy v bloku příkazů a cyklus se vrací opět k vyhodnocení podmínky. Takto se cyklus opakuje, dokud není podmínka vyhodnocena jako logicky nepravdivá, pak se cyklus ukončí.

Blok opakování obvykle slouží k úpravě proměnné, která se testuje v rámci podmínky příkazu. Představuje ho jeden či více příkazů, které se spouští při každém opakování.

Blok příkazů je seznam jednoho či více příkazů v hranatých závorkách, které se spouští pří každém opakování smyčky.

```
for ($i = 1; $i -le 10; $i++)
{
    Write-Host $i
}
```

Obr.23 Příklad příkazu for s deseti opakováními.

Zdroj: vlastní zpracování

V příkladu na obrázku se smyčka spouští tolikrát, dokud je hodnota proměnné $i menší nebo rovna 10. Spustí se tedy desetkrát.

Foreach

Příkaz foreach slouží k procházení všech položek v kolekci. Nejtypičtějším typem kolekce je pole. Pomocí příkazu foreach lze spustit jeden či více příkazů pro každou položku v poli.

```
foreach ($Položka in $Kolekce)
{
    <Blok příkazů>
}
```

Obr.24 Ukázka syntaxe příkazu foreach.

Zdroj: vlastní zpracování

Na obrázku lze vidět základní syntaxi příkazu foreach. Proměnná $Kolekce představuje kolekci pro iteraci. Proměnná $Položka představuje proměnnou, jejíž hodnotu PowerShell nastavuje na začátku každé iterace na další hodnotu v kolekci. Ve složených závorkách se nachází příkazy, které se mají provést pro každou iteraci.

```
$seznamDnů = "Pondělí","Úterý","Středa","Čtrvtek","Pátek"

foreach ($den in $seznamDnů)
{
    Write-Host $den
}
```

Obr.25 Příklad použití příkazu foreach.

Zdroj: vlastní zpracování

Na obrázku lze vidět příklad praktického použití příkazu foreach. Při prvním spuštění foreach se nastaví proměnná $den na první hodnotu z kolekce $seznamDnů, zde tedy “Pondělí“. Poté proběhne rutina Write-Host. Při dalším procházení smyčky je proměnná $den nastavena na další hodnotu z kolekce. Takto se to opakuje pro každou položku v kolekci.

While

Příkaz while slouží k vytvoření smyčky, která spouští příkazy v bloku příkazů tak dlouho, dokud je podmínka v závorkách logicky pravdivá.

```
while (<Podmínka>)
{
    <Blok příkazů>
}
```

Obr.26 Syntaxe příkazu while.

Zdroj: vlastní zpracování

Na obrázku lze vidět základní syntaxi příkazu while. V kulatých závorkách je podmínka, jejíž pravdivost se před každým průchodem vyhodnocuje. V hranatých závorkách se nachází příkazy, které se mají v každém cyklu opakovat.

Smyčka while vyhodnocuje pravdivost podmínky ještě před samotným spuštěním, pokud je hned na začátku vyhodnocena jako logicky nepravdivá, cyklus se nespustí ani jednou.

```
$hodnota = 0

while($hodnota -ne 3)
{
    $hodnota++
    Write-Host $hodnota
}
```

Obr.27 Příklad použití příkazu while.

Zdroj: vlastní zpracování

V následujícím příkladu cyklus while běží tak dlouho, dokud se proměnná $hodnota nerovná 3. Jakmile dosáhne hodnoty 3, podmínka se vyhodnotí jako nepravda a cyklus se ukončí.

Do

Příkaz do pracuje s klíčovým slovem while nebo until k cyklickému spouštění příkazů na základě vyhodnocení podmínky.

Do-While

```
do {<Blok příkazů>}
while (<Podmínka>)
```

Obr.28 Základní syntaxe příkazu do-while.

Zdroj: vlastní zpracování

Smyčka do-while funguje podobně jako smyčka while. Příkazy v bloku příkazů se cyklicky spouštějí, dokud je podmínka vyhodnocena jako logicky pravdivá. Jediným rozdílem je, že ve smyčce do-while se podmínka vyhodnocuje až po spuštění bloku příkazů. Blok příkazů se tedy vždy spustí minimálně jednou.

Do-Until

```
do {<Blok příkazů>}
until (<Podmínka>)
```

Obr.29 Základní syntaxe příkazu do-until.

Zdroj: vlastní zpracování

Na rozdíl od toho cyklus do-until spouští blok příkazů pouze tehdy, když je podmínka vyhodnocena jako logicky nepravdivá. Podobně jako v do-while se podmínka vyhodnocuje až po spuštění bloku příkazů, blok příkazů se tedy vždy spustí minimálně jednou.

Break a **Continue**

Příkazy break a continue nejsou přímo cykly, ale jejich využití se právě v cyklech uplatňuje.

Break

Příkaz break slouží k ukončení celého cyklu. Když se tento příkaz v cyklu objeví, PowerShell ho okamžitě ukončí a pokračuje na další příkazy pod tělem cyklu.

```powershell
$poleČísel = 1..10

foreach ($číslo in $poleČísel)
{
    if ($číslo -gt 5)
    {
        Write-Host "Číslo je větší než 5. Ukončuji cyklus."
        break
    }
}
```

Obr.30 Ukázka použití příkazu break.

Zdroj: vlastní zpracování

Na obrázku lze vidět ukázku použití příkazu break. Cyklus foreach prochází pole čísel, vnitřní podmínka kontroluje, zda aktuální číslo není větší než 5, pokud ano, cyklus se ukončí a foreach nepokračuje dalším procházením.

Break lze mimo všech cyklů použít i v příkazu switch

Continue

Příkaz continue slouží k ukončení aktuální iterace cyklu, nikoli ale celého cyklu. Jakmile se tento příkaz v cyklu objeví, PowerShell přeskočí zbytek kódu v aktuální iteraci, skočí na začátek cyklu a pokračuje iterací další.

```powershell
$poleČísel = 1..10

foreach ($číslo in $poleČísel)
{

    if ($číslo % 2 -eq 0)
    {
        continue
    }

    Write-Output "Lichá hodnota: $číslo"
}
```

Obr.31 Ukázka použití příkazu continue.

Zdroj: vlastní zpracování

Kód na příkladu v obrázku projede pole čísel pomocí cyklu foreach a vypíše pouze lichá čísla. Vnitřní podmínka if se spustí pouze v případě, když se objeví sudé číslo.

4.10.2 Pole

Pole je datová struktura sloužící k uložení kolekce více položek. Hodnoty uložené v poli mohou být stejného typu ale i různého datového typu.

K vytvoření pole stačí proměnné přiřadit několik hodnot oddělených čárkou.

```
$pole = 7,81,5,61,9,4,10
```

Obr.32 Ukázka syntaxe vytvoření pole.

Zdroj: vlastní zpracování

Na obrázku lze vidět základní syntaxi vytvoření jednoduchého pole se sedmi číslenými hodnotami.

```
$pole = 1..7
```

Obr.33 Ukázka vytvoření pole s použitím operátoru rozsahu.

Zdroj: vlastní zpracování

K vytvoření pole lze také použít operátor rozsahu (..).

Takto vytvořené pole bude obsahovat hodnoty 1,2,3,4,5,6,7.

Pokud při vytvoření nezadáme explicitně datový typ, PowerShell vytvoří pole jako kolekci objektů Systém.Object[]. K vytvoření pole určitého datového typu je třeba pole přetypovat zadáním konkrétního datového typu před název proměnné v hranatých závorkách.

```
[int32[]]$pole = 1..7
```

Obr.34 Syntaxe vytvoření silně typovaného pole.

Zdroj: vlastní zpracování

Takto vytvořené pole může obsahovat pouze hodnoty datového typu int32, tedy celá čísla.

```
$pole = @(1,2,3,4,5,6,7)
```

Obr.35 Ukázka vytvoření pole pomocí symbolu @.

Zdroj: vlastní zpracování

Druhým způsobem vytvoření pole je pomocí symbolu @.

Výhoda tohoto způsobu zápisu je ta, že takto lze vytvořit pole z příkazů. Výstup příkazů umístěných do kulatých závorek je automaticky umístěn do pole.

```
$pole = @(Get-Process)
```

Obr.36 Ukázka vytvoření pole z příkazu.

Zdroj: vlastní zpracování

V tomto případě se vytvoří pole z výstupu příkazu *Get-Process*.

Ke čtení a přístupu k prvkům pole stačí zadat název proměnné pole. Při zadání proměnné pole PowerShell vypíše všechny prvky v poli. K přístupu ke konkrétním prvkům v poli je třeba přistupovat pomocí indexu v hranatých závorkách.

```
$pole[0]
```

Obr.37 Ukázka syntaxe přístupu do pole pomocí indexu.

Zdroj: vlastní zpracování

Hodnoty indexu polí začínají na hodnotě 0. Na příkladu v obrázku lze tedy vidět přístup k první hodnotě v daném poli.

K přístupu k prvkům v poli lze také použít operátor rozsahu. Pro zobrazení více oblastí pole je možno použít operátor +.

```
$pole[0..6]

$pole[0,1+3..6]
```

Obr.38 Ukázka dalších možností přístupu pomocí indexu.

Zdroj: vlastní zpracování

Na vrchní ukázce lze vidět přístup k prvkům pole za pomocí operátoru rozsahu. PowerShell takto přistoupí k položkám s indexem 0,1,2,3,4,5,6.

Na spodní ukázce lze vidět přístup k více oblastem za pomocí operátoru +. PowerShell takto přistoupí k položkám s indexem 0,1 a 3,4,5,6.

Jedna z nejužitečnějších vlastností polí v PowerShellu je vlastnost Length případně Count. Ta vrací počet položek v poli.

```
$pole = 1,2,3,4,5,6,7

$pole.length
```

Obr.39 Ukázka použití vlastnosti Length.

Zdroj: vlastní zpracování

Příkaz v ukázce vrátí hodnotu 7.

4.10.3 Operátory porovnání

PowerShell podporuje základní aritmetické operátory pro provádění aritmetických operací. Jedná se o operátory sčítání (+), odečítání (-), násobení (*), dělení (/) a zbytek po dělení (%).

Mimo to PowerShell obsahuje operátory porovnání, které slouží k porovnání hodnot, případně k nalezení hodnoty, která odpovídá zadaným vzorům.

Operátor porovnání	Definice
-eq	Je rovno
-ne	Není rovno
-gt	Větší než
-ge	Větší než nebo rovno
-lt	Menší než
-le	Mešní než nebo rovno
-Like	Shoda se zástupným znakem
-NotLike	Neshoda se zástupným znakem
-Match	Shoda s regulárním výrazem
-NotMatch	Neshoda s regulárním výrazem
-Contains	Určuje, zda kolekce obsahuje zadanou hodnotu
-NotContains	Určuje, zda kolekce neobsahuje zadanou hodnotu
-In	Určuje, zda je zadaná hodnota v kolekci
-NotIn	Určuje, zda zadaná hodnota není v kolekci
-Replace	Nahradí zadanou hodnotu

Tabulka 1, Tabulka porovnávacích operátorů v PowerShellu.

Zdroj: Microsoft dokumentace [3]

Praktické použití operátorů -eq, -le, -ne, -gt bylo již v této práci ukázáno v rámci tématu podmínek a cyklů.

Operátor -Like provádí kontrolu shody se zástupnými znaky. Základní dva zástupné znaky v PowerShellu jsou „*“ a „?“. Zástupný znak „?“ zastupuje jeden znak v dané pozici. Zástupný znak „*“ slouží k zastoupení nula či více znaků.

```
'PowerShell' -like '*Shell'

'PowerShell' -like 'Power?hell'
```

Obr.40 Ukázka použití operátoru -like společně se zástupnými znaky.

Zdroj: vlastní zpracování

Na obrázku lze vidět ukázku dvou příkazů s operátorem porovnání -like a ukázku dvou zástupných znaků. Oba příkazy v ukázce jsou vyhodnoceny jako pravdivé a vrátí hodnotu „True“. Operátor -NotLike funguje obdobně akorát naopak.

Operátor -Match provádí kontrolu shody s regulárními výrazy. Regulární výrazy jsou vzory používané k porovnávání textu. Mohou se skládat ze znaků, operátorů a dalších konstrukcí.

Regulární výraz může být znak nebo řetězec znaků. Může zastupovat skupiny znaků pomocí hranatých závorek, případně rozsahy znaků. Pomocí znaku /d může zastupovat libovolné číslo. Umožňuje používat kvantifikátory. Umožňuje používat kotvy „^“ pro začátek řetězce a „$“ pro konec řetězce. Umožňuje vložit znaky do zpětných lomítek \...\ a ty pak nejsou analyzovány. Umožňuje používat znaky jako \t, což odpovídá kartě nebo znak \n, který odpovídá novému řádku a další znaky.

```
'test' -match 't[aeo]st'
```

Obr.41 Ukázka použití operátoru -match.

Zdroj: vlastní zpracování

Na ukázce v obrázku lze vidět použití operátoru -match. Jako regulární výraz jsou zde použity hranaté závorky, které specifikují skupinu přípustných znaků. Příkaz v ukázce je vyhodnocen jako pravdivý, jelikož písmeno „e“ se nachází ve skupině zástupných znaků a slova se tudíž shodují. Porovnání se slovem „tast“ nebo „tost“ by bylo též vyhodnoceno jako pravdivé.

Operátor -contains určuje, zda zadaná kolekce obsahuje zadanou hodnotu.

```
$pole = 1,2,4,5,6

$pole -contains 5
```

Obr.42 Ukázka praktického použití operátoru -contains.

Zdroj: vlastní zpracování

Příkaz v ukázce testuje, zda se číslo 5 nachází v kolekci $pole. To se v dané kolekci nachází a příkaz tedy správně vrací hodnotu „true“. Operátor -NotContains vrací opačný výsledek.

Operátor -in určuje zda se daná hodnota nachází v kolekci. Funguje podobně jako -contains akorát opačným směrem.

```
$pole = 1,2,4,5,6

5 -in $pole
```

Obr.43 Ukázka praktického použití operátoru -in.

Zdroj: vlastní zpracování

Na příkladu v obrázku lze vidět praktické použití operátoru -in. Příkaz je vyhodnocen jako pravdivý a vrací hodnotu „true“, stejně jako v předchozím příkladu. Operátor -NotIn vrací logicky opačnou hodnotu.

Operátor -replace slouží k nahrazení hodnoty jinou hodnotou.

```
$pole = 1,2,3

$pole -replace 2, 5
```

Obr.44 Ukázka syntaxe a použití operátoru -replace.

Zdroj: vlastní zpracování

V obrázku lze vidět syntaxi použití operátoru -replace. V tomto příkladu je v poli čísel hodnota 2 nahrazena hodnotou 5. Výsledné pole čísel poté obsahuje hodnoty 1,5,3. Místo kolekce hodnot lze také použít textový řetězec a nahradit jeho část.

5 Praktická část

Tato praktická část se zabývá tvorbou podpůrných video tutoriálů na téma správy Windows Serveru pomocí PowerShellu.

V rámci této praktické části je připraveno 7 úkolů, zaměřených na konkrétní oblasti/témata správy Windows Serveru.

K tomuto zadání je vždy přiložen písemný postup řešení daného úkolu, v podobě okomentovaného PowerShell skriptu a dále fotky řešení ke kontrole.

Dále je ke každému takovému úkolu zhotoven okomentovaný postup řešení ve formě video tutoriálu.

Okomentované PowerShell skripty včetně samotných video tutoriálů lze nalézt v přiložených souborech k této práci.

V rámci video tutoriálů je ukázán postup na Windows Server 2019 se statickou IP adresou 192.168.1.1 a maskou 255.255.255.0. Jméno serveru je nastaveno jako srv1.

Videa byla nahrána pomocí programu Open Broadcaster Software (OBS) a editována v programu PowerDirector 365.

Kvůli nedokonalému hlasovému projevu byla nakonec zvolena metoda předtáčení, kdy byl nejprve natočen samotný postup řešení a ten byl následně zpětně nadabován.

5.1 ActiveDirectory

Active Directory je adresářová služba od firmy Microsoft specifická pro systémy Windows. Běží na protokolu LDAP. Jedná se o hierarchickou strukturu, ve které jsou uchovávány informace o objektech v síti. Ukládají se zde například informace o objektech jako jsou počítače, tiskárny, uživatelské účty, skupiny apod. Ty lze dále dělit a organizovat do tzv. organizačních jednotek (OU). Jedním z hlavních využití Active Directory je ověřování uživatelů a počítačů v doméně a možnost aplikovat na ně systémové politiky. Když se hovoří o Active Directory, je nejčastěji myšlena role Windows Serveru Active Directory Domain Services (AD DS)

Zadání:

Nainstalujte roli Active Directory Domain Services a delegujte server na doménový řadič.

V případě potřeby vytvořte na disku nový oddíl pro databáze Active Directory.

Vytvořte organizační jednotku (OU) UHK a v ní OU Pocitace, Uzivatele, Skupiny.

V OU Skupiny vytvořte uživatelské skupiny Studenti, Ucitele, Administratori

V OU Uzivatele založte uživatele Jan novák, Petr Dvořák, Tomáš Černý, Eliška Novotná.

Nastavte jim uživatelský login skládající se z příjmení + prvních dvou písmen jejich jména (Tedy Jan Novák bude mít login novakja).

Nastavte uživatelům heslo, jehož platnost nikdy nevyprší a uživatel si ho nemusí změnit při prvním přihlášení.

Uživatele Tomáš Černý a Petr Dvořák přidejte do skupiny Studenti, uživatele Jan Novák do skupiny Ucitele a Elišku Novotnou do skupiny Administratori.

Do OU Pocitace přidejte PC1 a PC5.

Přidejte klientský PC do domény a zkuste se přihlásit na některý z účtů.

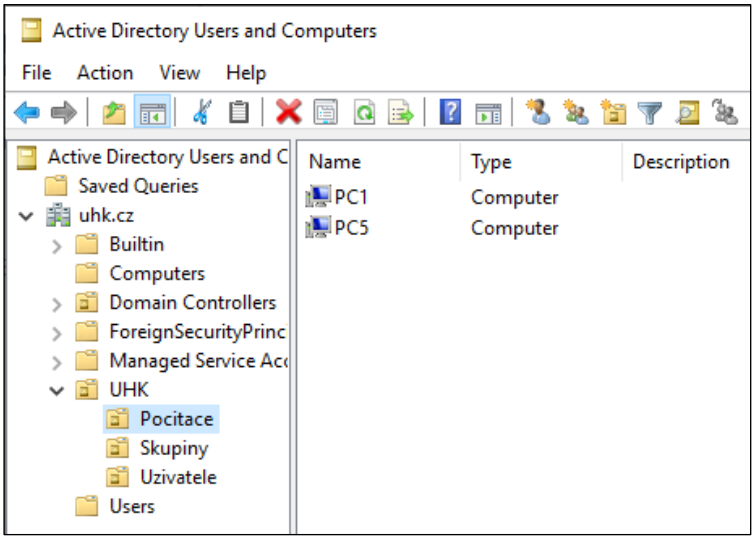
Řešení:

```
#Instalace role ADDS (ManagementTools bude sloužit pro
kontrolu)
Install-WindowsFeature -Name AD-Domain-Services -
IncludeManagementTools

#Kontrola disků v systému
Get-Disk

#Kontrola oddílů na zadaném disku
Get-Partition -DiskNumber 0

#Změna velikosti oddílu na zadanou hodnotu
Resize-Partition -DiskNumber 0 -PartitionNumber 1 -Size
90GB
#Vytvoření nového oddílu z veškerého nepřiděleného místa
na zadaném disku, označení písmenem F a naformátování
New-Partition -DiskNumber 0 -UseMaximumSize -DriveLetter F
| Format-Volume
```

```
#Instalace nového ActiveDirectory lesu s názvem "uhk.cz" a
uložení ActiveDirectory databáze včetně složek LogPath a
Sysvol na diskový oddíl F a instalace DNS. Zároveň
delegace serveru na doménový řadič.
Install-ADDSForest -DomainName "uhk.cz" -DatabasePath
"F:\Windows\NTDS" -LogPath "F:\Windows\NTDS" -SysvolPath
"F:\Windows\SYSVOL" -InstallDns:$true

#Vytvoření organizačních jednotek dle zadání.
New-ADOrganizationalUnit -Name "UHK" -Path "DC=UHK,DC=CZ"
New-ADOrganizationalUnit -Name "Uzivatele" -Path
"OU=UHK,DC=UHK,DC=CZ"
New-ADOrganizationalUnit -Name "Skupiny" -Path
"OU=UHK,DC=UHK,DC=CZ"
New-ADOrganizationalUnit -Name "Pocitace" -Path
"OU=UHK,DC=UHK,DC=CZ"

#Vytvoření uživatelských skupin
New-ADGroup -Name "Studenti" -GroupCategory 1 -GroupScope
1 -Path " OU=Skupiny,OU=UHK,DC=UHK,DC=CZ "
New-ADGroup -Name "Ucitele" -GroupCategory 1 -GroupScope 1
-Path " OU=Skupiny,OU=UHK,DC=UHK,DC=CZ "
New-ADGroup -Name "Administratori" -GroupCategory 1 -
GroupScope 1 -Path " OU=Skupiny,OU=UHK,DC=UHK,DC=CZ "

#Vytvoření uživatelů dle zadání
New-ADUser -Name "Jan Novák" -GivenName "Jan" -Surname
"Novák" -UserPrincipalName "novakja@uhk.cz" -
ChangePasswordAtLogon 0 -PasswordNeverExpires 1 -
AccountPassword (ConvertTo-SecureString -AsPlainText
"Jj123456" -Force) -Enabled 1 -Path
"OU=Uzivatele,OU=UHK,DC=UHK,DC=CZ"

New-ADUser -Name "Petr Dvořák" -GivenName "Petr" -Surname
"Dvořák" -UserPrincipalName "dvorakpe@uhk.cz" -
ChangePasswordAtLogon 0 -PasswordNeverExpires 1 -
AccountPassword (ConvertTo-SecureString -AsPlainText
"Pp123456" -Force) -Enabled 1 -Path
"OU=Uzivatele,OU=UHK,DC=UHK,DC=CZ"

New-ADUser -Name "Tomáš Černý" -GivenName "Tomáš" -Surname
"Černý" -UserPrincipalName "cernyto@uhk.cz" -
ChangePasswordAtLogon 0 -PasswordNeverExpires 1 -
AccountPassword (ConvertTo-SecureString -AsPlainText
"Tt123456" -Force) -Enabled 1 -Path
"OU=Uzivatele,OU=UHK,DC=UHK,DC=CZ"

New-ADUser -Name "Eliška Novotná" -GivenName "Eliška" -
Surname "Novotná" -UserPrincipalName "novotnael@uhk.cz" -
ChangePasswordAtLogon 0 -PasswordNeverExpires 1 -
AccountPassword (ConvertTo-SecureString -AsPlainText
"Ee123456" -Force) -Enabled 1 -Path
"OU=Uzivatele,OU=UHK,DC=UHK,DC=CZ"
```

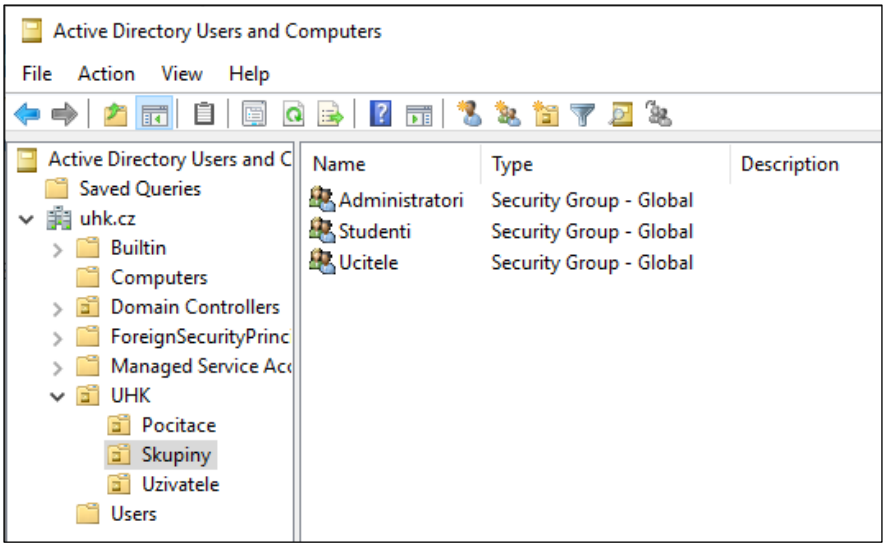
```
#Přidání uživatelů do skupin
Add-ADGroupMember -Identity Studenti -Members "Tomáš
Černý" , "Petr Dvořák"
Add-ADGroupMember -Identity Ucitele -Members "Jan Novák"
Add-ADGroupMember -Identity Administratori -Members
"Eliška Novotná"


#Přidání počítačů do organizační jednotky
New-ADComputer -Name "PC1" -Path
"OU=Pocitace,OU=UHK,DC=UHK,DC=CZ"
New-ADComputer -Name "PC5" -Path
"OU=Pocitace,OU=UHK,DC=UHK,DC=CZ"
```

Kontrola:



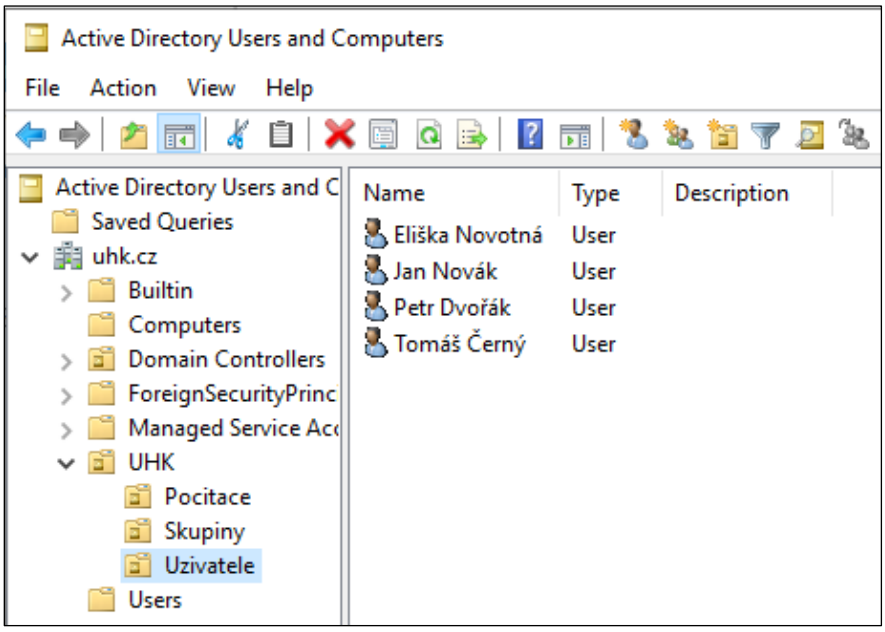
Obr.45 Ukázka správného nastavení Active Directory.

Zdroj: vlastní zpracování



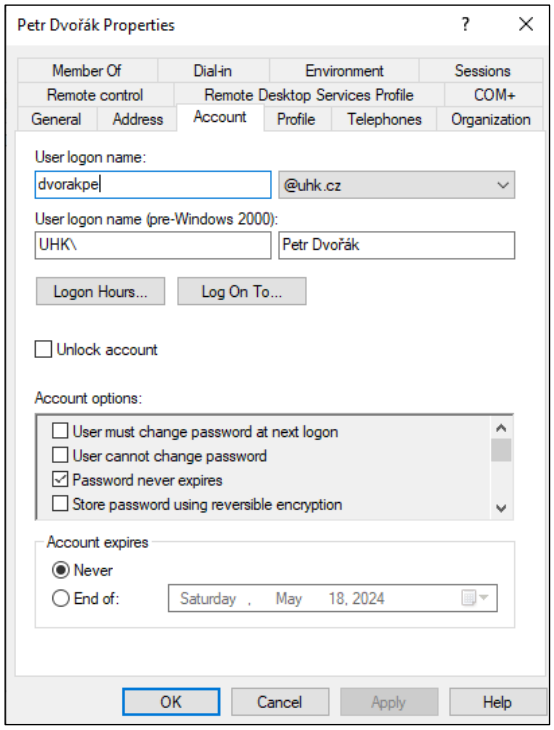
Obr.46 Ukázka správného nastavení Active Directory.

Zdroj: vlastní zpracování



Obr.47 Ukázka správného nastavení Active Directory.

Zdroj: vlastní zpracování



Obr.48 Ukázka správného nastavení uživatele v Active Directory.

Zdroj: vlastní zpracování

5.2 CSV

CSV (Comma Separated Values) je standardizovaný textový formát pro ukládání a zpracování tabulkových dat.

Každý řádek zde symbolizuje jednu položku a jednotlivé hodnoty v řádku jsou odděleny zvoleným oddělovačem (typicky čárka, nebo středník ;).

V oblasti správy Windows serveru se s využitím PowerShellu CSV soubory typicky používají pro automatizaci a jako jednoduchá možnost přidání obrovského množství dat, například ActiveDirectory objektů.

Zadání:

Vytvořte OU 'Absolventi' a naplňte ji daty z přiloženého CSV souboru data.csv.

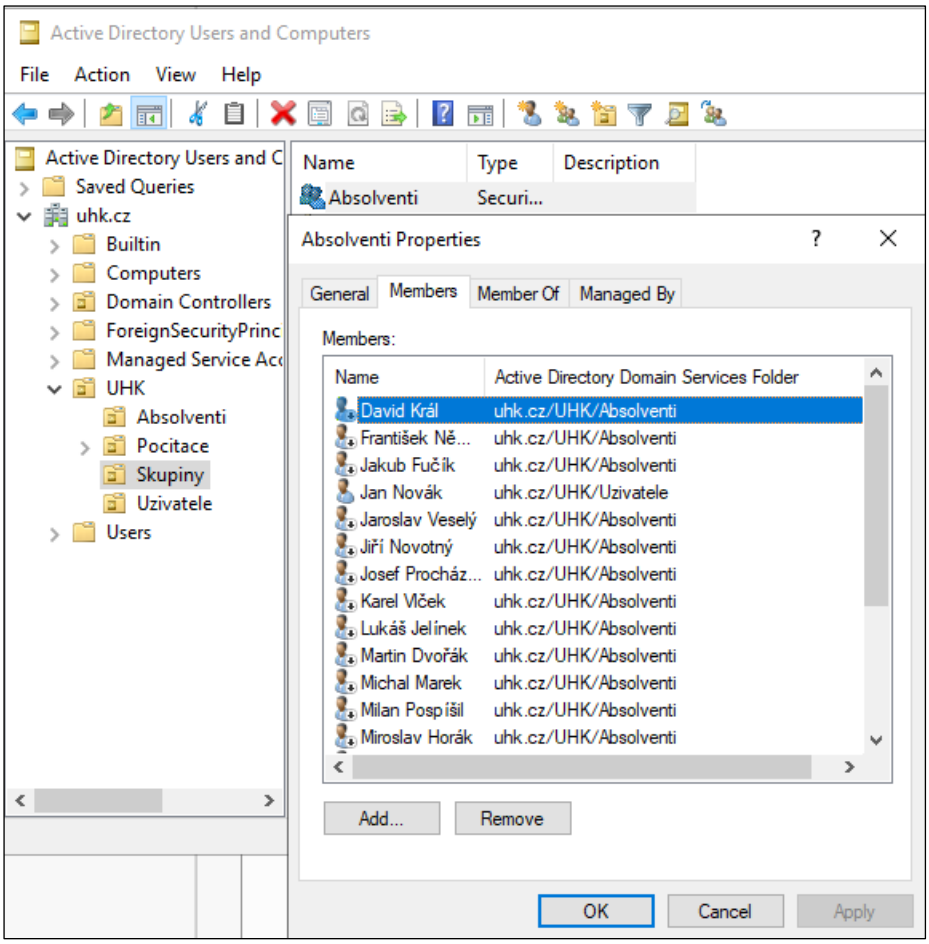
Vytvořte také stejnojmennou skupinu uživatelů a každého takového absolventa do ní přidejte.

Nastavte tyto uživatele tak aby neměli přístup do systému.

Řešení:

```
#Vytvoření organizační jednotky a skupiny dle zadání.
New-ADOrganizationalUnit -Name "Absolventi" -Path
"OU=UHK,DC=UHK,DC=CZ"
New-ADGroup -Name "Absolventi" -GroupCategory 1 -
GroupScope 1 -Path " OU=Skupiny,OU=UHK,DC=UHK,DC=CZ "


#Import CSV souboru a uložení do proměnné $data
$data = Import-Csv -Path
C:\Users\Administrator\Desktop\data.csv -Delimiter ";" -
Header 'jméno', 'příjmení'

#Foreach cyklus projde každý záznam a vytvoří nového
uživatele a přidá do skupiny
Foreach($person in $data)
{
 New-ADUser -Name ($person.jméno + " " + $person.příjmení)
-GivenName $person.jméno -Surname $person.příjmení -
Enabled 0 -Path "OU=Absolventi,OU=UHK,DC=UHK,DC=CZ"

 Add-ADGroupMember -Identity Absolventi -Members
($person.jméno + " " + $person.příjmení)
}
```

Kontrola:



Obr.49 Ukázka správně naimportovaných uživatelů ve skupině Absolventi.

Zdroj: vlastní zpracování

5.3 DNS

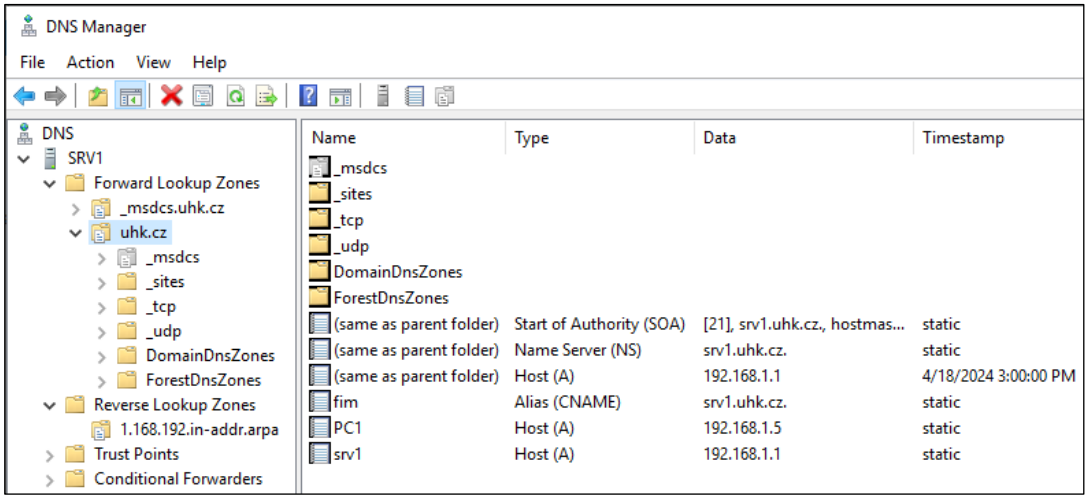
DNS (Domain Name System) je protokol, jehož základním principem je překlad IP adres na doménová jména a naopak.
Windows Server má pro tento protokol vlastní roli, kterou lze nainstalovat a konfigurovat buď pomocí GUI nebo pomocí PowerShellu.

Zadání:
Úkolem je nainstalovat roli DNS a nastavit její základní konfiguraci (pokud už máme nainstalováno ActiveDirectory, tak máme DNS již nainstalované a není třeba ho znovu instalovat).
Vytvořte dopřednou DNS zónu (Forward Lookup Zone) a související reverzní zónu (Reverse Lookup Zone) pro doménu uhk.cz. (pokud už máme nainstalováno ActiveDirectory, tak máme dopřednou zónu již hotovou).
Do této dopředné a reverzní zóny přidejte DNS Host záznam pro náš server ‘srv1‘ s adresou 192.168.1.1 a pro klientský počítač ‘PC1‘ s adresou 192.168.1.5, tak aby byly v síti dostupné pod určeným doménovým jménem.
Dále si pro vyzkoušení funkcionality DNS vytvořte alias 'fim', který bude odkazovat na náš server.
Poté proveďte kontrolu jak na serveru, tak na klientském PC připojeném k serveru – ověřte, zda nám DNS správně překládá doménová jména a adresy (Lze provést v PowerShellu případně v cmd). Nezapomeňte změnit DNS příponu počítače (Lze provést v PowerShellu, případně v GUI)

Řešení:

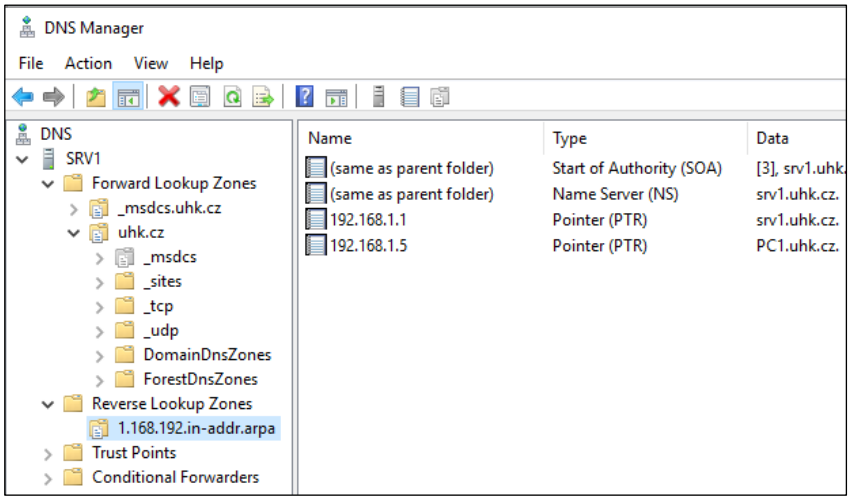
```
#Kontrola zda nemáme DNS už nainstalované
Get-WindowsFeature -Name DNS

#Instalace role DNS (Nutné pouze pokud nemáme
ActiveDirectory, ManagementTools bude sloužit pouze pro
následnou kontrolu)
Install-WindowsFeature -Name DNS -IncludeManagementTools

#Přidání primární dopředné/forward zóny
Add-DnsServerPrimaryZone -Name "uhk.cz" -ZoneFile
"uhk.cz.dns"
#Přidání primární reverzní/reverse zóny
Add-DnsServerPrimaryZone -NetworkID 192.168.1.0/24 -
ZoneFile "1.168.192.in-addr.arpa.dns"

#Vytvoření DNS host záznamu pro server a pro klientský PC
Add-DnsServerResourceRecordA -Name "srv1" -ZoneName
"uhk.cz" -AllowUpdateAny -IPv4Address "192.168.1.1" -
CreatePtr
Add-DnsServerResourceRecordA -Name "PC1" -ZoneName
"uhk.cz" -AllowUpdateAny -IPv4Address "192.168.1.5" -
CreatePtr


#Vytvoření aliasu, který odkazuje na náš server
Add-DnsServerResourceRecordCName -Name "fim" -
HostNameAlias "srv1.uhk.cz" -ZoneName "uhk.cz"

#Změna DNS přípony v registrech na uhk.cz
Set-ItemProperty -Path
"HKLM:\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters"
-Name "Domain" -Value "uhk.cz"

#Kontrola v PowerShellu zda nám DNS správně překládá
doménová jména na správné adresy
Resolve-DnsName pc1
Resolve-DnsName fim

#Kontrola zda nám DNS správně překládá adresy na doménová
jména
Resolve-DnsName -Name 192.168.1.5 -Type PTR
```

Kontrola:



Obr.50 Ukázka správně nastavené dopředné zóny.

Zdroj: vlastní zpracování



Obr.51 Ukázka správně nastavené reverzní zóny.

Zdroj: vlastní zpracování

```
Name                                           Type   TTL   Section    IPAddress
----                                           ----   ---   -------    ---------
pc1.uhk.cz                                     A      3600  Answer     192.168.1.5

Name       : fim.uhk.cz
QueryType  : CNAME
TTL        : 3600
Section    : Answer
NameHost   : srv1.uhk.cz

srv1.uhk.cz                                    A      3600  Answer     192.168.1.1

Name       : 5.1.168.192.in-addr.arpa
QueryType  : PTR
TTL        : 3600
Section    : Answer
NameHost   : PC1.uhk.cz
```

Obr.52 Ukázka konzole PowerShellu, kde DNS správně překládá doménová jména na IP adresy a také naopak.

Zdroj: vlastní zpracování

5.4 DHCP

DHCP (Dynamic Host Configuration Protocol) je TCP/IP protokol, který automaticky přiděluje hostům v síti IP adresu a další související síťovou konfiguraci jako například masku sítě, výchozí bránu nebo adresu DNS serveru.
Ve Windows Serveru tento protokol zastupuje role DHCP.

Zadání:

Nainstalujte roli DHCP.

Vytvořte security groups.

V případě že máte ActiveDirectory, DHCP autorizujte.

Smažte upozornění v ServerManageru (volitelné)

Povolte dynamické aktualizace DNS

Vytvořte DHCP obor s názvem 'uhkScope' s rozmezím adres 192.168.1.10-192.168.1.200 a s dobou zapůjčení 1 den.

Nastavte v tomto oboru výchozí bránu a adresu DNS serveru.

Nastavte výjimku na adresy 192.168.1.100-192.168.1.110, které se nebudou hostům přidělovat.

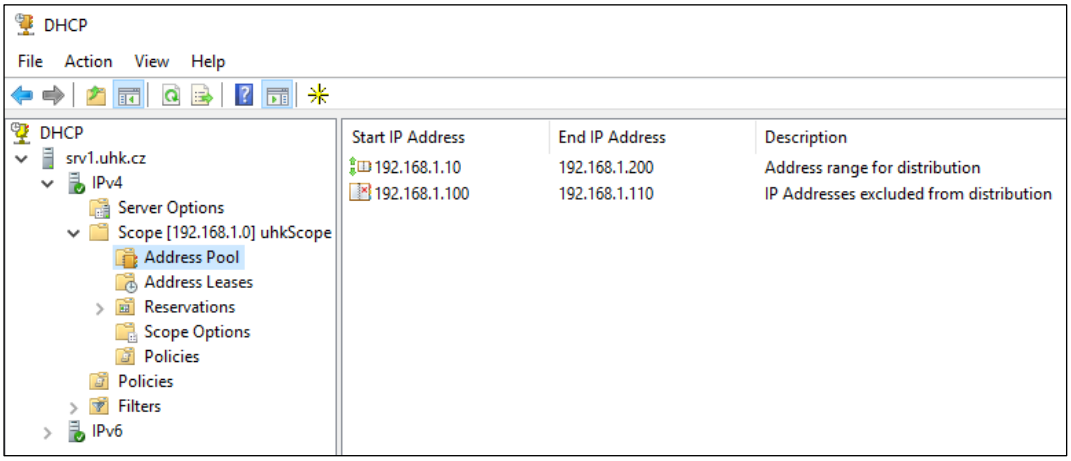
Udělte rezervaci IP adresy 192.168.1.50 pro klienta s MAC adresou 94:DE:80:5D:12:FA s popiskem 'Rezervace pro tiskárnu'

Přihlaste se na klientský PC, nastavte automatické získávání IP konfigurace a zkontrolujte, zda mu byla správně přidělena.

Řešení:

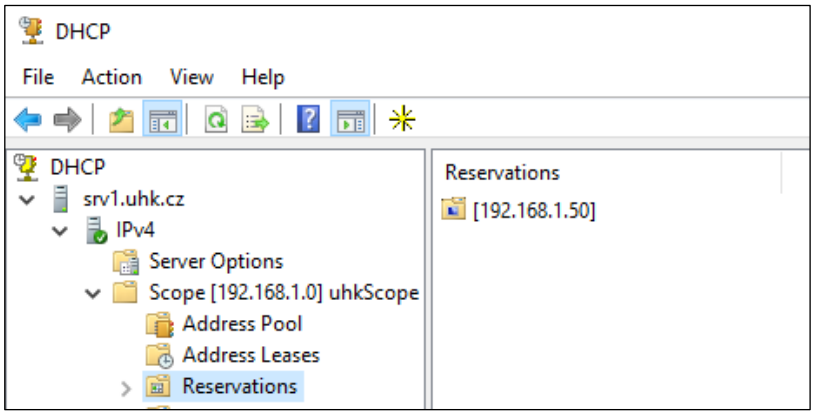
```
#Instalace role DHCP (rozhraní ManagementTools pro
kontrolu)
Install-WindowsFeature DHCP -IncludeManagementTools

#Vytvoření security groups a restartování služby DHCP
Netsh DHCP Add SecurityGroups
Restart-Service dhcpserver

#Autorizace DHCP v AD
Add-DHCPServerInDC -DnsName srv1.uhk.cz -IPAddress
192.168.1.1
#Kontrola
Get-DhcpServerInDC

#Smazání upozornění v server manageru
Set-ItemProperty –Path
registry::HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\ServerMana
ger\Roles\12 –Name ConfigurationState –Value 2

#DNS dynamické aktualizace
Set-DhcpServerv4DnsSetting -ComputerName "srv1.uhk.cz" -
DynamicUpdates "Always"

#Vytvoření DHCP oboru dle zadání s dobou zápůjčky 1 den
Add-DhcpServerv4Scope -name “uhkScope” -StartRange
192.168.1.10 -EndRange 192.168.1.200 -SubnetMask
255.255.255.0 -State Active -LeaseDuration 1:00:00:00


#Nastavení výchozí brány a DNS serveru
Set-DhcpServerv4OptionValue -ScopeID 192.168.1.0 -
DnsServer 192.168.1.1
Set-DhcpServerv4OptionValue -ScopeID 192.168.1.0 -Router
192.168.1.1

#Udělení výjimky na rozsah adres 192.168.1.100-
192.168.1.110
Add-DhcpServerv4ExclusionRange -ScopeID 192.168.1.0 -
StartRange 192.168.1.100 -EndRange 192.168.1.110

#Rezervace IP pro konkrétního klienta
Add-DhcpServerv4Reservation -ScopeId 192.168.1.0 -
IPAddress 192.168.1.50 -ClientId "94-DE-80-5D-12-FA" -
Description "Rezervace pro tiskárnu"
```

Kontrola:



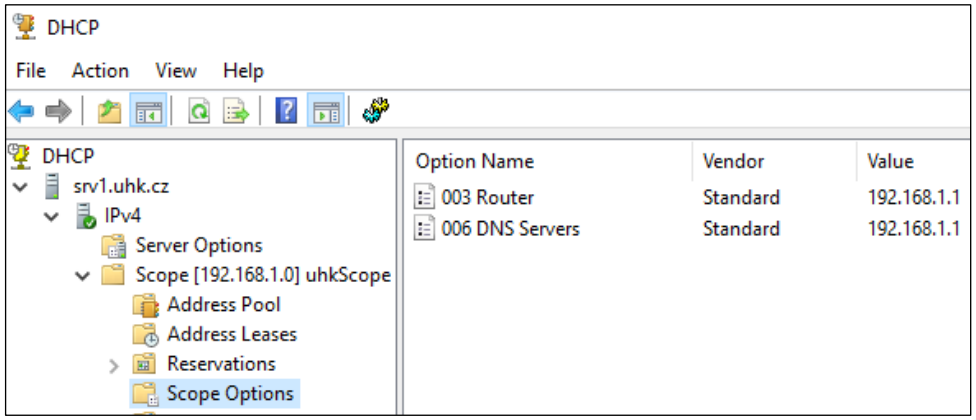
Obr.53 Ukázka správně nastaveného DHCP oboru.

Zdroj: vlastní zpracování



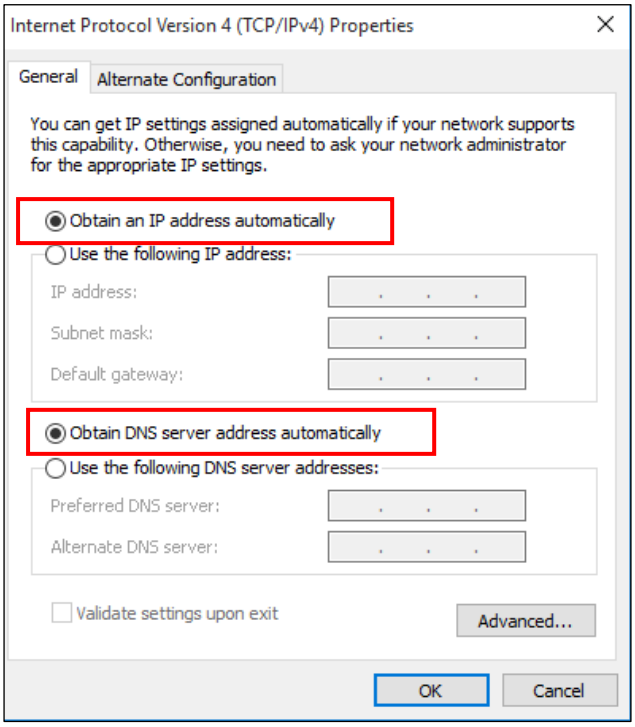
Obr.54 Ukázka správně nastavené rezervace.

Zdroj: vlastní zpracování



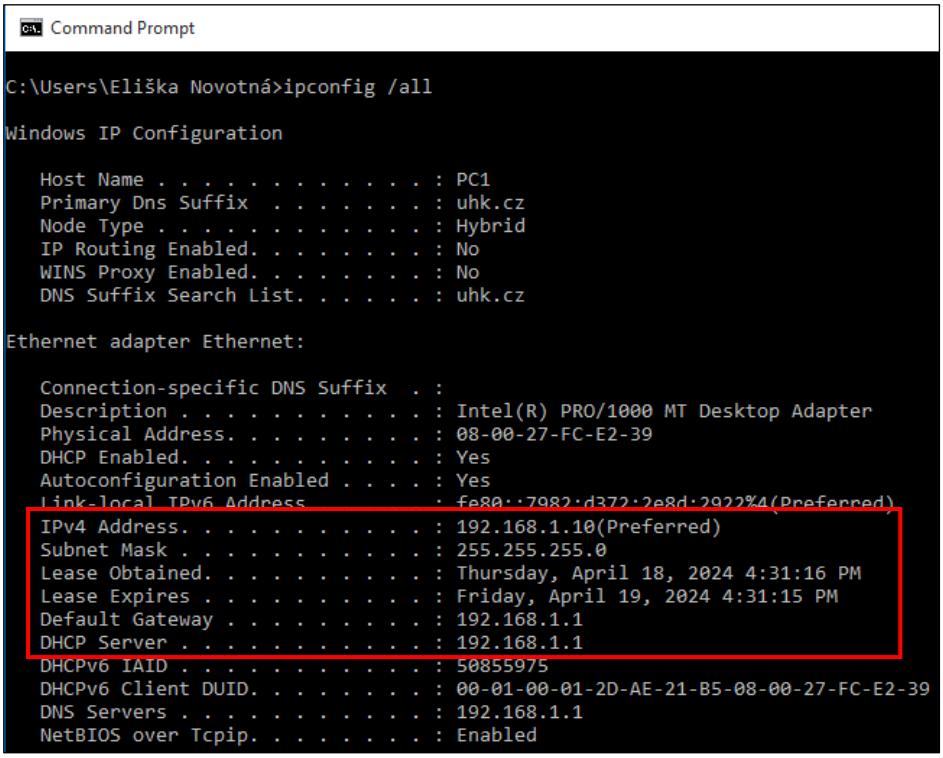
Obr.55 Ukázka správně nastavené výchozí brány a adresy DNS serveru.

Zdroj: vlastní zpracování



Obr.56 Nastavení automatického získávání IP konfigurace na klientovi.

Zdroj: vlastní zpracování



Obr.57 Kontrola správně přidělené IP konfigurace.

Zdroj: vlastní zpracování

5.5 Sdílená domovská složka (SMB + ACL)

SMB (Server Message Block) je síťový protokol, který umožňuje sdílení souborů po síti. Umožňuje uživatelům a aplikacím přistupovat k souborům (ale i jiným zdrojům) na vzdáleném serveru. Je to jeden ze způsobů, jak implementovat domovské složky pro uživatele v doméně.

ACL (Access Control List) je ve Windows seznam bezpečnostních oprávnění připojených k určitému objektu, zejména složkám a souborům. Tyto pravidla určují, kdo může k danému objektu přistupovat a jaké operace s ním může provádět.

Zadání:

Vytvořte a nasdílejte uživatelům v doméně (studentům a učitelům) jejich domovské složky.

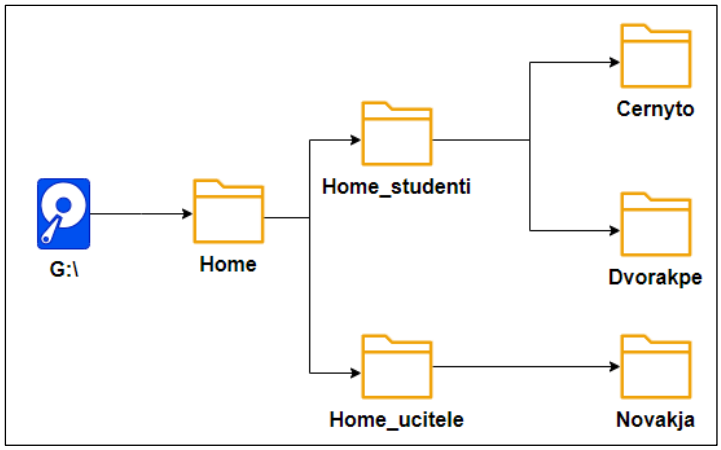
Nejprve vytvořte nový disk (nový oddíl) pro tyto domovské složky. Přidělte mu písmeno G.

Na tomto disku vytvořte složku Home a v této složce dále složky Home_studenti a Home_ucitele.

Ve složce Home_studenti vytvořte každému studentovi složku s jeho login jménem (tedy Cernyto a Dvorakpe).

V Home_ucitele vytvořte domovskou složku Novakja pro učitele Jana Nováka.

Následná mapa složek by vypadala následovně:



Obr.58 Mapa složek na disku G:

Zdroj: vlastní zpracování v https://app.diagrams.net/

Tento disk a složky v něm zabezpečte podle následujících pravidel:
Uživatelé Administrators a SYSTEM budou mít přístup k disku a všem jeho složkám a podsložkám s plným oprávněním (FullControl).
Uživatelé ověření v doméně (Authenticated Users) budou mít k disku, složce Home a složce Home_studenti a Home_ucitele přístup ke čtení.
K uživatelským domovským složkám bude mít (kromě výše uvedených) přístup pouze konkrétní daný uživatel (tedy například k domovské složce Cernyto bude mít přístup pouze student Tomáš Černý a účet Administrators a SYSTEM).
Tito uživatelé budou mít ve svých domovských složkách plná práva kromě smazání své domovské složky, měnění práv a přebírání vlastnictví.

Nasdílejte složku Home pod názvem "Share" a písmenem G a nastavte skupině Administrators plný přístup k tomuto sdílení a skupině Authenticated Users právo měnit.
Poté namapujte uživatelům v ActiveDirectory jejich domovské složky.

Řešení:

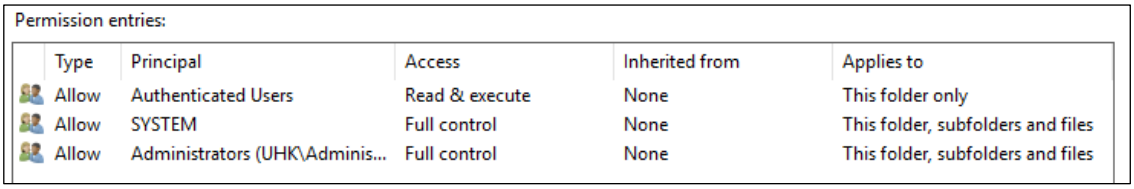
```
#Vytvoření nového diskového oddílu G
Get-Disk
Get-Partition -DiskNumber 0
Resize-Partition -DiskNumber 0 -PartitionNumber 1 -Size
80GB
New-Partition -DiskNumber 0 -UseMaximumSize -DriveLetter G
| Format-Volume

#Vytvoření složek dle zadání
New-Item -ItemType "directory" -Path "G:\" -Name "Home"
New-Item -ItemType "directory" -Path "G:\Home\" -Name
"Home_Studenti"
New-Item -ItemType "directory" -Path "G:\Home\" -Name
"Home_Ucitele"
New-Item -ItemType "directory" -Path
"G:\Home\Home_Studenti\" -Name "Cernyto"
New-Item -ItemType "directory" -Path
"G:\Home\Home_Studenti\" -Name "Dvorakpe"
New-Item -ItemType "directory" -Path
"G:\Home\Home_Ucitele\" -Name "Novakja"

#Zabezpečení G:\
#Načteme ACL
$acl = Get-Acl „G:\“
```

```powershell
#Smažeme existující pravidla v ACL
$acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) |
Out-Null }

#Vytvoříme jednotlivá pravidla dle zadání
$adminRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Admini
strators", "FullControl", "ContainerInherit,
ObjectInherit", "None", "Allow")
$systemRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("SYSTEM
", "FullControl", "ContainerInherit, ObjectInherit",
"None", "Allow")
$authRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Authen
ticated Users", "Read", "None", "None", "Allow")

#A přidáme je do ACL
$acl.AddAccessRule($adminRule)
$acl.AddAccessRule($systemRule)
$acl.AddAccessRule($authRule)

#Nastavíme ACL zpět na objekt
Set-Acl -Path "G:\" -AclObject $acl


#Zabezpečení Home
$acl = Get-Acl -Path "G:\Home\"
$acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) |
Out-Null }
$authRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Authen
ticated Users", "Read", "None", "None", "Allow")
$acl.AddAccessRule($authRule)
Set-Acl -Path "G:\Home\" -AclObject $acl

#Zabezpečení Home_studenti
$acl = Get-Acl -Path "G:\Home\Home_studenti\"
$acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) |
Out-Null }
$studentiRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Studen
ti", "Read", "None", "None", "Allow")
$acl.AddAccessRule($studentiRule)
Set-Acl -Path "G:\Home\Home_studenti\" -AclObject $acl

#Zabezpečení Home_ucitele
$acl = Get-Acl -Path "G:\Home\Home_ucitele\"
$acl.Access | ForEach-Object { $acl.RemoveAccessRule($_) |
Out-Null }
$uciteleRule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Ucitel
e", "Read", "None", "None", "Allow")
$acl.AddAccessRule($uciteleRule)
Set-Acl -Path "G:\Home\Home_ucitele\" -AclObject $acl
```

```
#Zabezpečení domovské složky pro učitele Jana Nováka
$acl = Get-Acl -Path "G:\Home\Home_ucitele\Novakja"
$acl.Access | ForEach-Object {$acl.RemoveAccessRule($_) |
Out-Null}

#Vytvoříme speciální oprávnění. Ze skupiny práv Modify
bitovým operátorem -bxor odebereme právo mazat tuto složku
a bitovým operátorem -bor přidáme možnost mazat podsložky
a soubory
$specialRights =
[System.Security.AccessControl.FileSystemRights]::Modify
-bxor
[System.Security.AccessControl.FileSystemRights]::Delete
-bor
[System.Security.AccessControl.FileSystemRights]::DeleteSu
bdirectoriesAndFiles

$rule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Jan
Novák", $specialRights, "ContainerInherit,
ObjectInherit","None", "Allow")
$acl.AddAccessRule($rule)
Set-Acl -Path "G:\Home\Home_ucitele\Novakja" -AclObject
$acl

#Zabezpečení domovské složky pro studenta Tomáše Černého
$acl = Get-Acl -Path "G:\Home\Home_studenti\Cernyto"
$acl.Access | ForEach-Object {$acl.RemoveAccessRule($_) |
Out-Null}
$rule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Tomáš
Černý", $specialRights, "ContainerInherit,
ObjectInherit","None", "Allow")
$acl.AddAccessRule($rule)
Set-Acl -Path "G:\Home\Home_studenti\Cernyto" -AclObject
$acl

#Pro studenta Petra Dvořáka
$acl = Get-Acl -Path "G:\Home\Home_studenti\Dvorakpe"
$acl.Access | ForEach-Object {$acl.RemoveAccessRule($_) |
Out-Null}
$rule = New-Object
System.Security.AccessControl.FileSystemAccessRule("Petr
Dvořák", $specialRights, "ContainerInherit,
ObjectInherit","None", "Allow")
$acl.AddAccessRule($rule)
Set-Acl -Path "G:\Home\Home_studenti\Dvorakpe" -AclObject
$acl
```

```
#Vytvoření SMB sdílení složky G:\Home pod názvem share s
práv dle zadání
New-SmbShare -Name "share" -Path "G:\Home" -FullAccess
"Administrators" -ChangeAccess "Authenticated Users"

#Namapování uživatelům v ActiveDirectory jejich domovských
složek pod písmenem G
Set-ADUser -Identity "Eliška Novotná" -HomeDirectory
"\\srv1\share\" -HomeDrive "G"
Set-ADUser -Identity "Tomáš Černý" -HomeDirectory
"\\srv1\share\Home_Studenti\Cernyto" -HomeDrive "G"
Set-ADUser -Identity "Petr Dvořák" -HomeDirectory
"\\srv1\share\Home_Studenti\Dvorakpe" -HomeDrive "G"
Set-ADUser -Identity "Jan Novák" -HomeDirectory
"\\srv1\share\Home_Ucitele\Novakja" -HomeDrive "G"
```

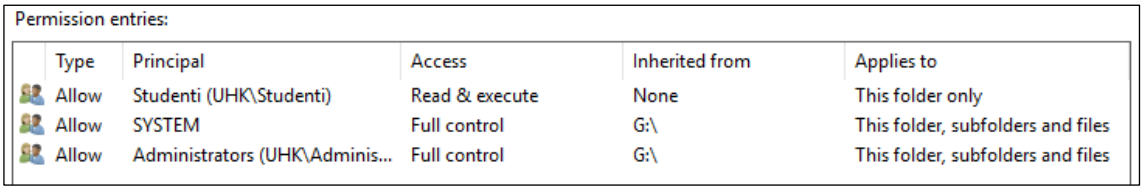
Kontrola:


Permission entries:

Type	Principal	Access	Inherited from	Applies to
Allow	Authenticated Users	Read & execute	None	This folder only
Allow	SYSTEM	Full control	None	This folder, subfolders and files
Allow	Administrators (UHK\Adminis...	Full control	None	This folder, subfolders and files

Obr.59 Ukázka nastavení oprávnění na disku G:\ (stejné nastavení je i na složce Home

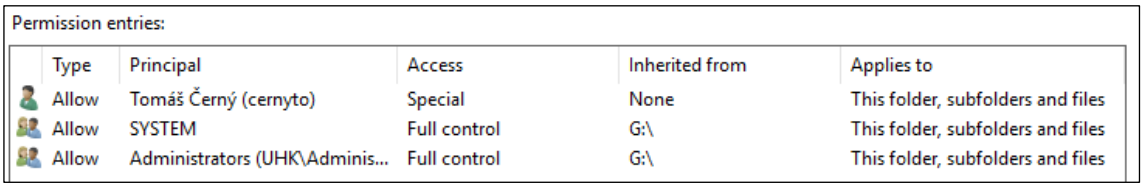
Zdroj: vlastní zpracování


Permission entries:

Type	Principal	Access	Inherited from	Applies to
Allow	Studenti (UHK\Studenti)	Read & execute	None	This folder only
Allow	SYSTEM	Full control	G:\	This folder, subfolders and files
Allow	Administrators (UHK\Adminis...	Full control	G:\	This folder, subfolders and files

Obr.60 Ukázka nastavení oprávnění na složce Home_studenti (analogicky pro Home_ucitele)

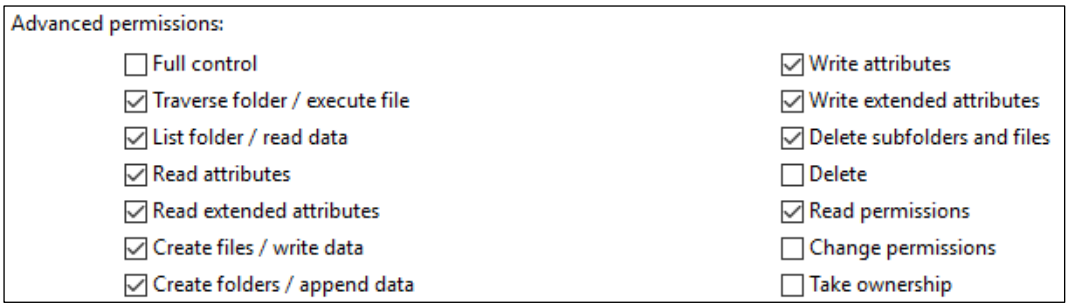
Zdroj: vlastní zpracování


Permission entries:

Type	Principal	Access	Inherited from	Applies to
Allow	Tomáš Černý (cernyto)	Special	None	This folder, subfolders and files
Allow	SYSTEM	Full control	G:\	This folder, subfolders and files
Allow	Administrators (UHK\Adminis...	Full control	G:\	This folder, subfolders and files

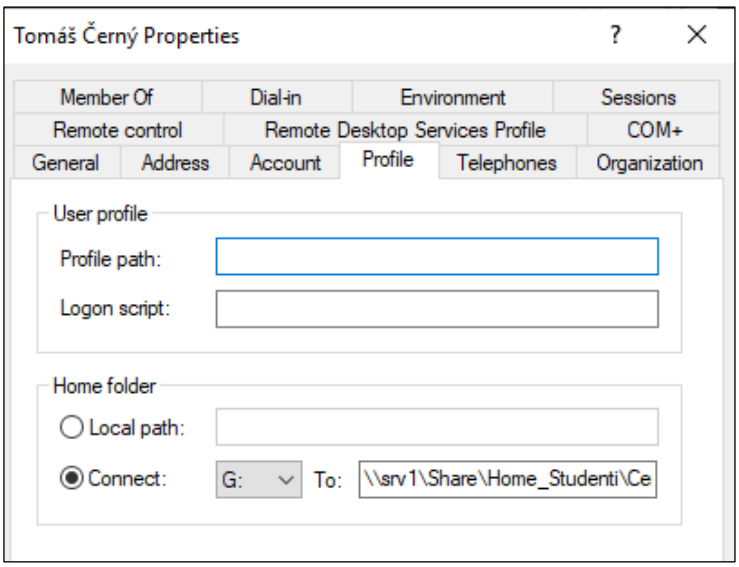
Obr.61 Ukázka nastavení oprávnění na domovské složce Cernyto (analogicky pro ostatní uživatele)

Zdroj: vlastní zpracování



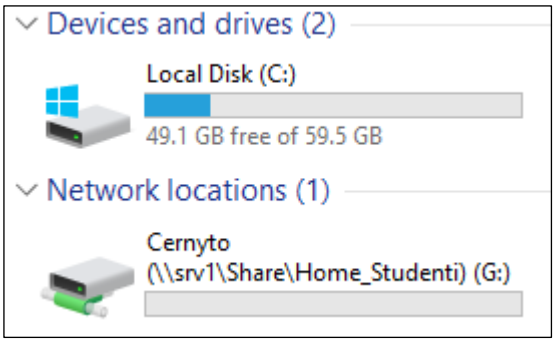
Obr.62 Ukázka speciálních oprávnění pro uživatele na jejich domovské složce.

Zdroj: vlastní zpracování



Obr.63 Ukázka nastavení namapování domovské složky v ActiveDirectory.

Zdroj: vlastní zpracování



Obr.64 Kontrola na klientském PC, kde je úspěšně připojena domovská složka jako disk G:

Zdroj: vlastní zpracování

5.6 GPO

Group Policy (Skupinové Politiky) je ve Windows nástroj, který umožňuje hromadné nastavení prostředí pro uživatelské účty a počítače. Největší uplatnění lze nalézt s využitím ActiveDirectory, kdy je jednoduše umožněno aplikovat politiky na velké množství počítačů/uživatelů v doméně. Group Policy lze ale využít i bez ActiveDirectory a spravovat tak lokální politiky na samostatných počítačích. Kolekce skupinových politik se označuje jako GPO – Group Policy Object (Objekty Skupinové Politiky). Nejčastěji tyto politiky fungují na principu úprav jednotlivých registrů a linkují se na organizační jednotky v ActiveDirectory.

Zadání:

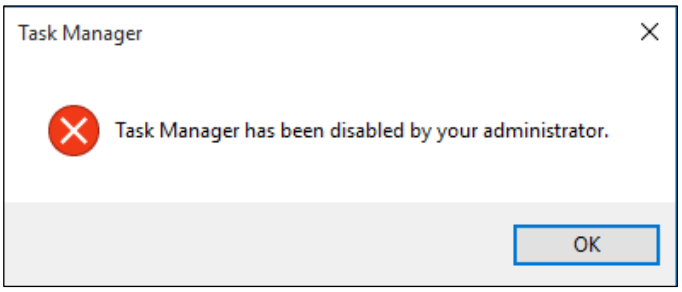
Vytvořte novou politiku GPO s názvem "Ucitele Configuration", která změní uživatelům ve skupině "Ucitele" jejich pozadí plochy.

Vytvořte druhou politiku GPO s názvem "Studenti Configuration", která uživatelům ve skupině "Studenti" zakáže používat příkazovou řádku (CMD) a správce úloh (Task Manager).

Řešení:

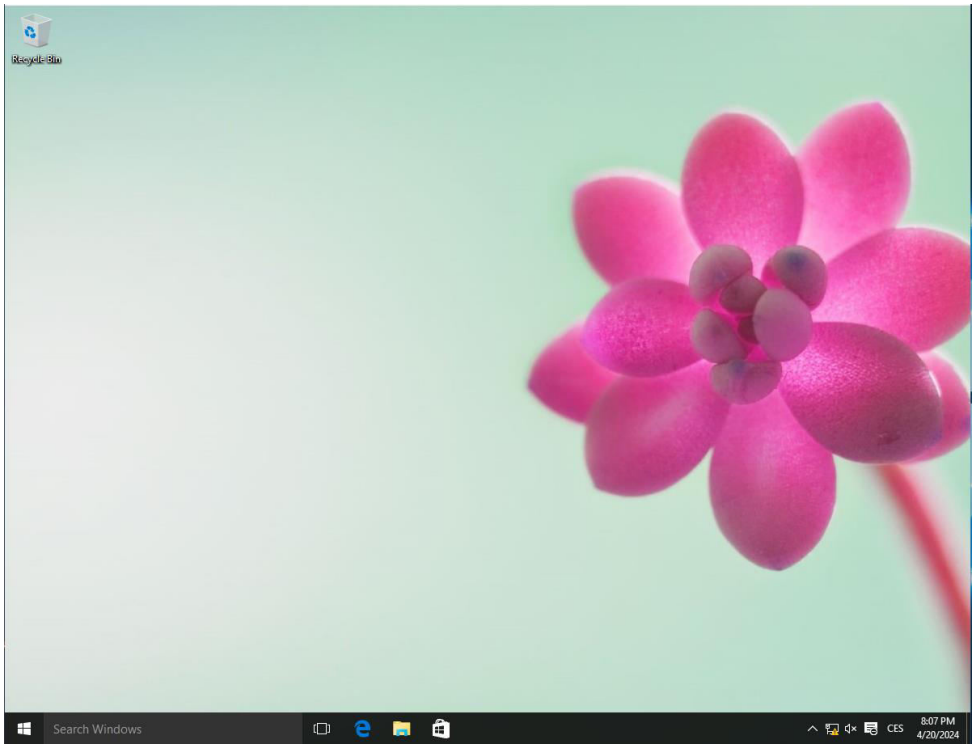
```
#Změna pozadí pro učitele
#Vytvoření nového GPO
New-GPO -Name "Ucitele Configuration"

#Nastavení aby politika neplatila pro skupinu
Authenticated Users, bez tohoto nastavení by politika
platila na všechny uživatele v doméně
Set-GPPermission -Name "Ucitele Configuration" -
PermissionLevel None -TargetName "Authenticated Users" -
TargetType Group -Replace

#Nastavení aby politika platila pro skupinu Ucitele
Set-GPPermission -Name "Ucitele Configuration" -
PermissionLevel GpoApply -TargetName "Ucitele" -TargetType
Group

#Nastavení registru pro pozadí plochy na jedno z výchozích
možných pozadí ploch systému Windows v cestě
C:\Windows\Web\Wallpaper\Theme2
Set-GPregistryValue -Name "Ucitele Configuration" -Key
"HKCU\Control Panel\Desktop" -ValueName "Wallpaper" -Type
String -Value "C:\Windows\Web\Wallpaper\Theme2\img9.jpg"

#Nalinkování GPO na OU Uzivatele
New-GPLink -Name "Ucitele Configuration" -target
"OU=Uzivatele,OU=UHK,DC=UHK,DC=CZ"



#Zákaz CMD a Task managera pro studenty
New-GPO -Name "Studenti Configuration"

Set-GPPermission -Name "Studenti Configuration" -
PermissionLevel None -TargetName "Authenticated Users" -
TargetType Group -Replace
Set-GPPermission -Name "Studenti Configuration" -
PermissionLevel GpoApply -TargetName "Studenti" -
TargetType Group

#Nastavení hodnoty v registru, které zakáže použití CMD
Set-GPregistryValue -Name "Studenti Configuration" -Key
"HKCU\Software\Policies\Microsoft\Windows\System" -
ValueName "DisableCMD" -Type DWORD -Value 2
#Nastavení hodnoty v registru, které zakáže použití Task
Managera
Set-GPRegistryValue -Name "Studenti Configuration" -Key
"HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\S
ystem" -ValueName "DisableTaskMgr" -Type DWORD -Value 1

New-GPLink -Name "Studenti Configuration" -target
"OU=Uzivatele,OU=UHK,DC=UHK,DC=CZ"
```

Kontrola:



Obr.65 Kontrola spuštění Správce úloh na studentském účtu.

Zdroj: vlastní zpracování



Obr.66 Kontrola na učitelském účtu, kde je po přihlášení změněno pozadí plochy.

Zdroj: vlastní zpracování

5.7 PowerShell Remoting

Ve verzi PowerShell 2.0 byla přidána funkce Remoting, která umožňuje vzdálenou správu počítačů, respektive spouštění příkazů z lokálního počítače na vzdáleném počítači ve stejné síti.

Zadání:

1) Povolte vzdálenou správu a připojte se ke vzdálenému PC v doméně. Vytvořte u něj na ploše soubor test.txt a zkontrolujte.

2) (Volitelné) Připojte se ke vzdálenému PC mimo doménu a otestujte

Řešení:

1)

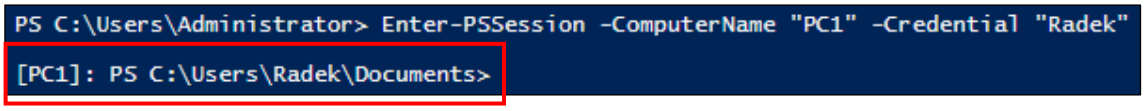
```
#Na PC, které chceme spravovat musíme nejprve povolit
vzdálenou správou, dialogové okno potvrdíme -> Yes to all
Enable-PSRemoting

#Ze serveru se připojíme k PC, které chceme spravovat
Enter-PSSession PC1

#Pro otestování funkčnosti připojení si zkusíme vytvořit
soubor na ploše PC, které spravujeme
New-Item -Path "C:\Users\Petr Dvořák\Desktop\text.txt" -
ItemType "File"

#Po otestování se ze vzdálené správy odpojíme
Exit-PSSession
```

2)

```
#Vzdálenou správu nyní musíme povolit na obou PC, jak na klientském PC, které budeme spravovat, tak i na serveru. Abychom se vyhnuli eroru kvůli veřejné síti, použijeme parametr -SkipNetworkProfileCheck
Enable-PSRemoting -SkipNetworkProfileCheck -Force

#Poté musíme přidat spravovaný PC do skupiny TrustedHosts
Set-Item WSMan:\localhost\Client\TrustedHosts "PC1"  -Force

#Kontrolu lze provést příkazem
Get-Item WSMan:\localhost\Client\TrustedHosts

#Ze serveru se připojíme k PC. Kromě názvu PC, ke kterému se chceme připojit, musíme zadat i název lokálního účtu pod kterým se chceme připojit
Enter-PSSession -ComputerName PC1 -Credential "Radek"

#Pro otestování funkčnosti připojení si zkusíme vytvořit soubor na ploše
New-Item -Path "C:\Users\Radek\Desktop\text.txt" -ItemType "File"

#Po otestování se ze vzdálené správy odpojíme
Exit-PSSession
```

Kontrola:

Úspěšné připojení lze poznat podle změněného řádku v konzoli.



Obr.67 Ukázka konzole po připojení ke vzdálenému PC.

Zdroj: vlastní zpracování

6 Shrnutí výsledků

Hlavním praktickým výsledkem práce je 7 cvičení zaměřených na různé oblasti správy Windows Serveru za pomocí PowerShellu a souvisejících skriptů a video tutoriálů.

Za osobní výsledek získaný v průběhu práce bych označil prohloubení znalostí z oblasti PowerShellu ale také z oblasti operačních systémů Windows a Windows Server a dále také z oblasti tvorby a editace videí.

7 Závěry a doporučení

V závěru práce naplnila má očekávání a povedlo se mi dle mého názoru, jak formou teoretickou, tak formou praktickou, ukázat vhodnost a sílu PowerShellu ke správě Windows Serveru.

Praktickou část bych ohodnotil jako zdařilou, ale se svým hlasovým projevem ve videích nejsem plně spokojen, hlavně kvůli občasnému koktání.

Jako další možná doporučení bych zmínil:

-Možnost doplnění videí o titulky, zejména pro neslyšící, případně překlad pro cizince

Praktická cvičení s video tutoriály by se daly dále rozvíjet o další témata a oblasti jako jsou například:

-Konfigurace služby IIS

-Auditování událostí v doméně

-Mapování disků pomocí GPO

-Konfigurace role Hyper-V

-Konfigurace Windows Deployment Services (WDS)

-Konfigurace Remote Desktop Services (RDS)

-Konfigurace Windows Server Update Services (WSUS)

8 Seznam použité literatury

[1] PowerShell. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-11-21]. Dostupné z: https://cs.wikipedia.org/wiki/PowerShell

[2] KAŠNÝ, Vojtěch, HÉL, Samuel, ed. Itnetwork. PowerShell - Online kurz [online]. 2020 [cit. 2023-11-15]. Dostupné z: https://www.itnetwork.cz/windows/powershell

[3] MICROSOFT. PowerShell Documentation. Microsoft.com [online]. [cit. 2023-11-15]. Dostupné z: https://learn.microsoft.com/en-us/powershell/

[4] SCHWICHTENBERG, Holger. WINDOWS POWERSHELL 5 UND POWERSHELL 7 [online]. 4. Carl Hanser Verlag GmbH & Co., 2020 [cit. 2023-11-20]. ISBN 978-3-446-45913-7. Dostupné z: https://www.hanser-elibrary.com/doi/10.3139/9783446460812

[5] CBS News. CBS News Monthly Poll #3 [online]. Inter-university Consortium for Political and Social Research, 2007 [cit. 2023-11-20]. Dostupné z: https://doi.org/10.3886/ICPSR21921.v1

[6] PowerShell. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2023-11-21]. Dostupné z: https://en.wikipedia.org/wiki/PowerShell

[7] MICROSOFT. Approved Verbs for PowerShell Commands [online]. 2022 [cit. 2023-11-25]. Dostupné z: https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/approved-verbs-for-windows-powershell-commands?view=powershell-7.4

[8] MICROSOFT. Strongly Encouraged Development Guidelines [online]. [cit. 2023-11-25]. Dostupné z: https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/strongly-encouraged-development-guidelines?view=powershell-7.4

[9] MICROSOFT CORPORATION. PowerShell Gallery [online]. [cit. 2023-12-04]. Dostupné z: https://www.powershellgallery.com/

[10] MICROSOFT. How to write a PowerShell module manifest [online]. [cit. 2023-12-04]. Dostupné z: https://learn.microsoft.com/en-us/powershell/scripting/developer/module/how-to-write-a-powershell-module-manifest?view=powershell-7.3

[11] MICROSOFT. PowerShell is replacing Command Prompt [online]. [cit. 2023-12-14]. Dostupné z: https://support.microsoft.com/en-us/windows/powershell-is-replacing-command-prompt-fdb690cf-876c-d866-2124-21b6fb29a45f

[12] MICROSOFT. Announcing Windows 10 Insider Preview Build 14971 for PC [online]. 2016 [cit. 2023-12-14]. Dostupné z: https://blogs.windows.com/windows-insider/2016/11/17/announcing-windows-10-insider-preview-build-14971-for-pc/

[13] PowerShell is Microsoft's latest open source release, coming to Linux, OS X. Arstechnica.com [online]. 2016 [cit. 2023-12-14]. Dostupné z: https://arstechnica.com/information-technology/2016/08/powershell-is-microsofts-latest-open-source-release-coming-to-linux-os-x/

[14] GITHUB. PowerShell [online]. [cit. 2023-12-14]. Dostupné z: https://github.com/PowerShell/PowerShell

[15] Proč PowerShell. DevBlog [online]. 2012 [cit. 2024-01-29]. Dostupné z: https://devblog.cz/2012/04/proc-powershell/

[16] What is PowerShell and how to use it: The ultimate tutorial. Techtarget [online]. [cit. 2024-01-29]. Dostupné z: https://www.techtarget.com/searchwindowsserver/definition/PowerShell

9 Přílohy

Veškeré přílohy jsou dostupné na přiloženém flash disku a online na OneDrivu pod následujícím odkazem: https://unihk-my.sharepoint.com/:f:/g/personal/dusekra1_uhk_cz/EoPTa5oBU6BIlBIehT1yjNsBvEOyvSwIzk7KNRq28be_UA

Videotutoriály:

1) Active Directory.mp4
2) CSV.mp4
3) DNS.mp4
4) DHCP.mp4
5) SMB + ACL.mp4
6) GPO.mp4
7) Remoting.mp4

PowerShell skripty:

1) Active Directory.ps1
2) CSV.ps1
3) DNS.ps1
4) DHCP.ps1
5) SMB+ACL.ps1
6) GPO.ps1
7) Remoting.ps1

Csv soubory:

1) data.csv

Příloha č. 1

Oskenované zadání práce

Zadání bakalářské práce

Autor:	**Radek Dušek**
Studium:	I2000471
Studijní program:	B0688A140001 Informační management
Studijní obor:	Informační management
Název bakalářské práce:	**Správa Windows serveru za využití PowerShell**
Název bakalářské práce AJ:	

Cíl, metody, literatura, předpoklady:

Cílem bakalářské práce je vytvořit podpůrné materiály v oblasti správy Windows serveru za využití Power Shell v podobě video tutoriálů. V teoretické části autor představí a podrobně popíše postupy využití Power Shellu. V praktické části pak autor vytvoří řešení dílčích úloh správy Windows serveru s využitím Power Shell ve formě video tutoriálů.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Josef Horálek, Ph.D.

Datum zadání závěrečné práce: 15.10.2021