# PowerShell

Róbert Novotný

robert.novotny@upjs.sk

# Skriptovanie

- práca Windows je primárne v grafickom rozhraní (GUI)
  - prívetivé, jednoduché
  - zvládne to aj BFU („bežný Fero-užívateľ“)
  - „windowsáci sú klikači“
- GUI však nestačí

# GUI nestačí!

- automatizácia úloh sa bije s filozofiu GUI
- Dilema: ako zautomatizovať klikanie?
- Dilema 2: prečo automatizovať klikanie?

# Skriptovanie v operačných sytémoch

- skriptovanie v OS umožňuje programovaním zautomatizovať administrátorské úkony
- skriptovanie = programovanie

# Skriptovacie jazyky pre OS

- používané už od nepamäti
  - od čias, keď GUI prakticky neexistovalo
  - a ani nebolo potrebné
- shell = command line interpreter
  - interpreter príkazov z riadku

# Nudná a dramatická história shellov

- 1961: **RUNCOM** [op. sys. CTSS]
  - možnosť spúšťať jednoduché príkazy s parametrami
- 1965: op. sys. **Multics**
  - presmerovanie vstupov a výstupov príkazov
  - pomocou samostatných príkazov

# Nudná a dramatická história shellov

- 1971: **Thompson Shell**
  - presmerovanie súčasťou syntaxe: príkazy **>** a **|**
  - základné elementy programového toku implementované príkazmi (if, goto)

# Nudná a dramatická história shellov

- 1977: **Bourne shell** (/bin/sh)
  - pridané elementy štruktúrovaného programovania
  - for, if, vyhodenie goto
  - premenné (netypované)
  - premenné prostredia
  - volanie podskriptov

# Nudná a dramatická história shellov

- 1987-...: **Bash** (GNU Project)
  - de facto štandardný shell v Linuxe
  - nadmnožina príkazov /bin/sh
  - ďalšie dodatočné vlastnosti prebraté z iných používaných shellov

# Nudná a dramatická história shellov

- 1981-2000: **command.com**
  - MS-DOS, Windows 9X a ME
  - bežné operácie sú priamo zabudované
    - dir, cd, mkdir
  - štruktúrované programovanie biedne
    - FOR, GOTO, IF, CALL
  - prakticky na úrovni roku 1971 ☹

# cmd.exe (1999+)

- Windows 2000, XP, Vista, 7
- niekoľko rozšírení oproti command.com
  - vylepšený FOR
  - vylepšený IF
- ale stále tragédia oproti UNIXovým shellom
- programovanie skriptov stále na úrovni 70. rokov

# Windows Script Host (1999+)

- skriptovacie prostredie
- možnosť voľby skriptovacieho jazyka
  - VBScript, JScript, Visual Basic,
  - možnosť doinštalovať ďalšie
- objektovo orientovaný prístup

# Windows Script Host (1999+)

- objektovo orientovaný prístup
- integrácia s COM
  - Component Object Model
    - vo Windowse starší spôsob zverejňovania objektov použiteľných krížom cez aplikácie
- problém: WSH sa akosi neujal

# Windows PowerShell (2006+)

- shell + plnoprávny skriptovací jazyk
  - od 2.0 súčasťou Win7 a Win2008 R2
    - stiahnuteľné do XP a novších
  - od sept. 2012 verzia 3.0
    - stiahnuteľné do Win7 SP1, Win2008 R2 SP1, Win2008 SP2
- silná integrácia s .NET
  - ako alternatíva k COM
- kompatibilita s cmd.exe

# Klasický shell = text

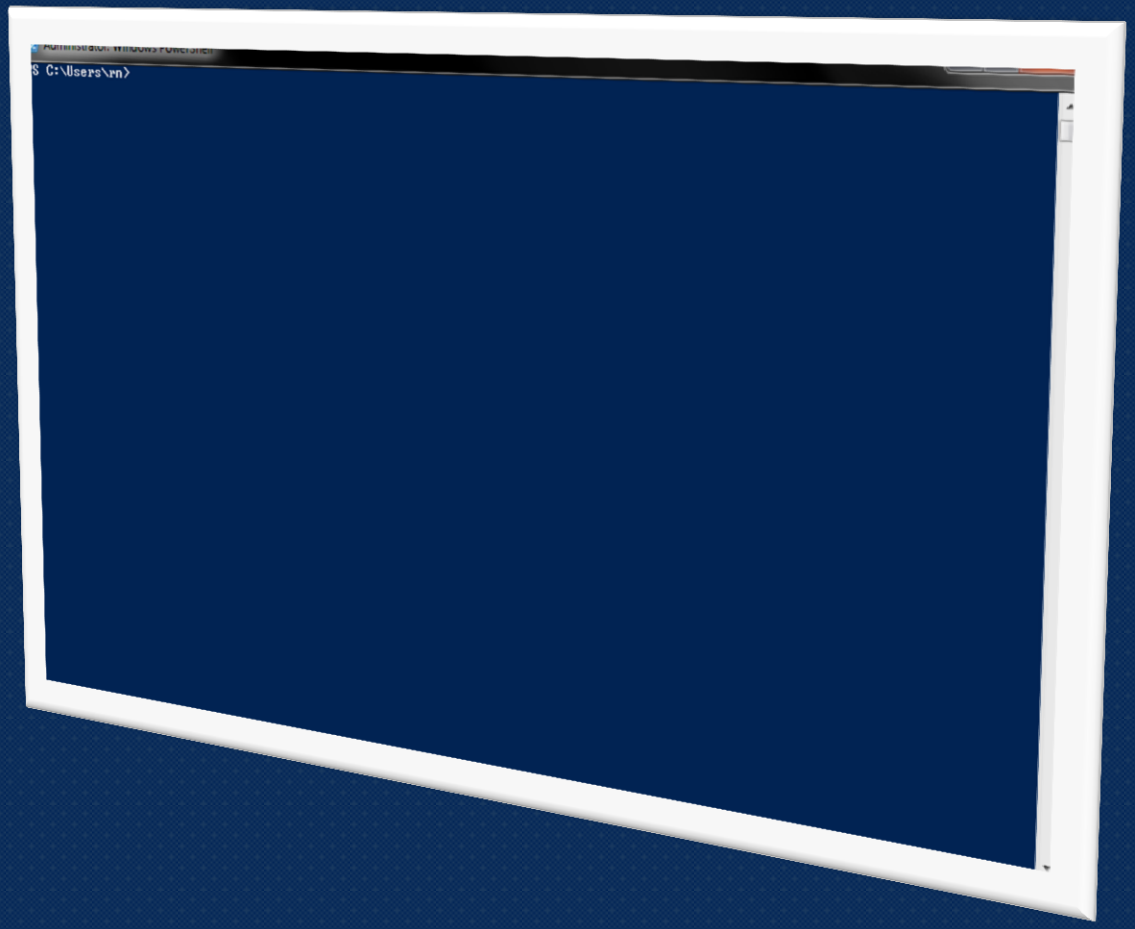
- klasické unixovské shelly sú **textovo orientované**
  - príkazy žujú a pľujú text
  - pri dobrej vôli je ako-tak štruktúrovaný...
- ...ale líši sa to od príkazu k príkazu
  - položky sú oddelené medzerou (ls)
  - položky sú oddelené tabulátorom (ls –la)
  - položky sú oddelené dvojbodkou (/etc/passwd)
- štandardný repertoár: **cut**, **awk**

# PowerShell = objekty

- silná objektová orientácia
- v PowerShelli lietajú hore-dole objekty
  - používateľ vidí ich textové reprezentácie
- properties (vlastnosti) = inštančné premenné = objekty majú stav
- metódy (schopnosti) = objekty majú samostatné schopnosti, ktoré vedia vykonať

# Úvod do PowerShell

- Windows 7: Start | powershell.exe



# Úvodné príkazy

- na prvý pohľad veľmi podobné klasickému cmd.exe
- command prompt: výzva pre príkaz
- obsahuje názov aktuálneho adresára
  - implicitne začíname v domovskom adresári

# PowerShell ako kalkulačka

```
PS C:\Users\rn> 2+2
4
```

- jednoduché výpočty fungujú priamo
- operátory: +, -, *, / (desatinné delenie), % (mod)
- matematické funkcie však nehľadajte

# Premenné

```
PS C:\Users\rn> $dph = 0.19
PS C:\Users\rn> $dph
0,19
```

- názvy premenných začínajú dolárom
- priradenie cez =
- výpis premennej: stačí napísať jej názov
- alternatívne: **echo $dph**

# Čo možno spustiť v PowerShelli

- spustiteľné programy: **notepad.exe**
- **cmdlets**
  - komplexné príkazy dostupné v .NET knižnici
  - zabudovaných množstvo cmdletov
  - vyvíjať nové nie je problém
- **funkcie**
  - jednoduchšie postupnosti príkazov s parametrami
- **PowerShell skripty**
  - viacriadkové postupnosti príkazov

# Zoznam vyvolateľných príkazov

```
Get-Command
```

- kilometrový zoznam obsahujúci
  - cmdlety
  - funkcie
  - aliasy: alternatívne názvy niektorých cmdletov
- samotný **Get-Command** je cmdlet

# Cmdlety v PowerShelli

- každý cmdlet má svoje **meno**
- silne dodržiavaná konvencia:

```
[sloveso]-[podstatné meno]
```

- **Get**-Cmdlet, **Set**-Content, **Stop**-Service...

# Aliasy

- množstvo cmdletov má svoje aliasy
- alternatívne názvy
- skracujú zápisy
- uľahčujú prechod z iných shellov

```
Get-Alias
```

```
alias
```

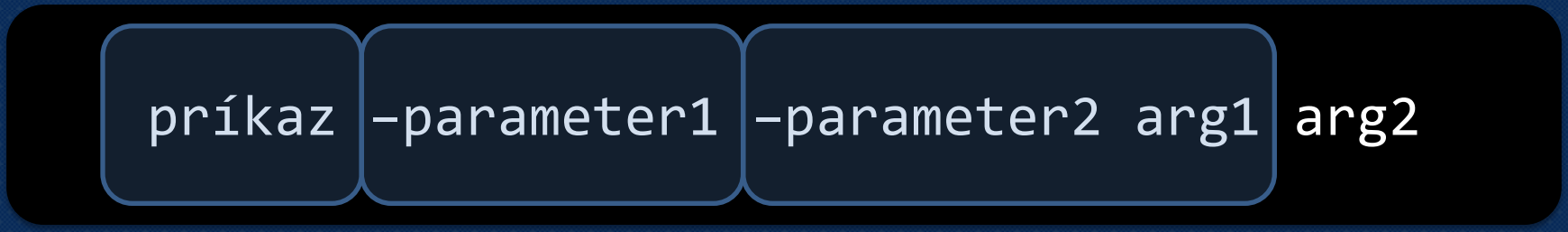
**alias** je alias pre Get-Alias

# Starí známi z DOSu a *nixu

| | | | |
|---|---|---|---|
| help | man | Get-Help | pomoc |
| dir | ls | Get-ChildItem | výpis adresárov a súborov v aktuálnom adresári |
| md | mkdir | (funkcia) | vytvorí nový adresár |
| cd | | Set-Location | presun o adresár vyššie/nižšie |
| del | rm | Remove-Item | odstránenie súboru / adresára |
| C:, D: ... | | (funkcie) | zmena aktuálnej jednotky |

# Príkazy, parametre a argumenty

- Príkaz pozostáva
  - z názvu príkazu
  - z prepínačov (parametrov)
    - vždy uvedené jednou pomlčkou
  - parameter môže mať argumenty
  - pozičných argumentov



# Jednoduchý príklad

- **help dir**: (návod k príkazu **dir**)
  - názov príkazu: **help**
  - žiadne parametre
  - jeden pozičný argument
- **dir -?** (význam ten istý)
  - názov príkazu: **dir**
  - jeden parameter
  - žiadne pozičné argumenty

alternatívne **man**
**ls**

# Cmdlety generujú objekty

`dir`

## unixovská mentalita

- ls získa zoznam súborov
- pošle ho na ďalšie spracovanie v podobe **textu**
  - položky oddelené medzerou
  - samostatné položky na nových riadkoch

## powershellovská mentalita

- ls získa zoznam súborov
- pošle ho na ďalšie spracovanie ako **sadu objektov typu súbor**
  - súbor má vlastnosti
    - meno, veľkosť...
  - a schopnosti
    - kopíruj, zmaž..

# Cmdlety sú jednoúčelové

- unixovská filozofia: príkaz rieši jednu vec a rieši ju najlepšie ako vie
- komplexné úlohy riešime skladaním viacerých príkazov
- cmdlet zbiera dáta, spracuje ich a pošle na ďalšie spracovanie

# Rúra - pipe

- rúra = **objekto**vod
- výstupom cmdletu je zoznam objektov
- cmdlet posiela do rúry objekt za objektom
- na rúru môžu byť napojené ďalšie cmdlety
  - prijmú objekt z rúry, spracujú, pošlú ďalej

# Rúra - pipe

- objekty prúdia postupnosťou cmdletov
  - v Unixe: vzniká kolóna príkazov
- možno prepájať cmdlety
- výstup z jedného cmdletu je zaslaný na vstup iného cmdletu
  - "prerúrovaný"
  - "piped"

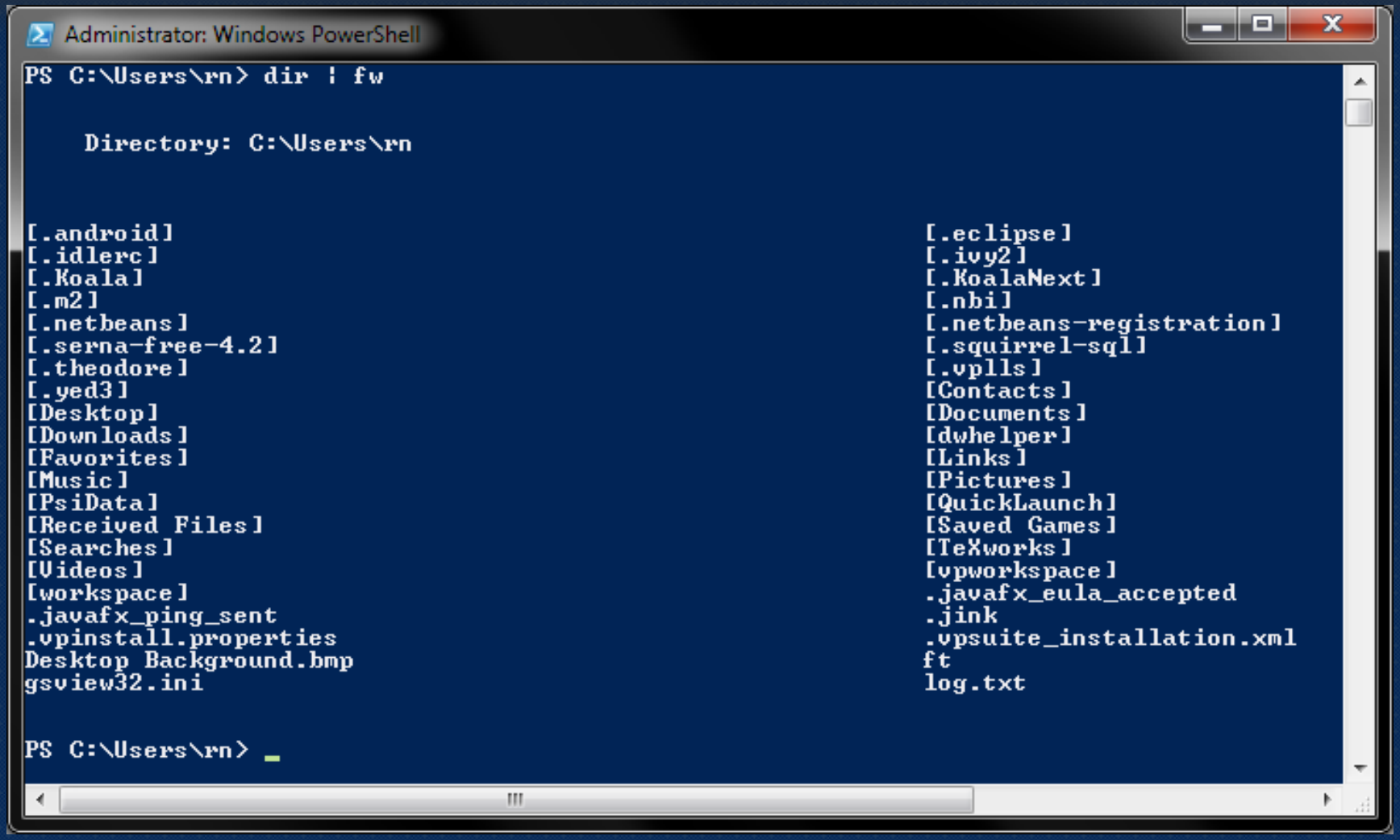
# Príklad rúry

```
dir | fw
```

fw = format wide

- **dir** vygeneruje zoznam objektov
- **fw** naformátuje zoznam objektov do širokej tabuľky, v ktorej sa zobrazí jeden atribút (property) objektu
  - property = atribút = inštančná premenná z OOP
- znak **|** (pipe, rúra) vytvorí rúru

# Výstup rúry


Administrator: Windows PowerShell
PS C:\Users\rn> dir | fw
Directory: C:\Users\rn
[.android]
[.eclipse]
[.idlerc]
[.ivy2]
[.Koala]
[.KoalaNext]
[.m2]
[.nbi]
[.netbeans]
[.netbeans-registration]
[.serna-free-4.2]
[.squirrel-sql]
[.theodore]
[.vplls]
[.yed3]
[Contacts]
[Desktop]
[Documents]
[Downloads]
[dwhelper]
[Favorites]
[Links]
[Music]
[Pictures]
[PsiData]
[QuickLaunch]
[Received Files]
[Saved Games]
[Searches]
[TeXworks]
[Videos]
[vpworkspace]
[workspace]
.javafx_eula_accepted
.javafx_ping_sent
.jink
.vpinstall.properties
.vpsuite_installation.xml
Desktop Background.bmp
ft
gsview32.ini
log.txt
PS C:\Users\rn>

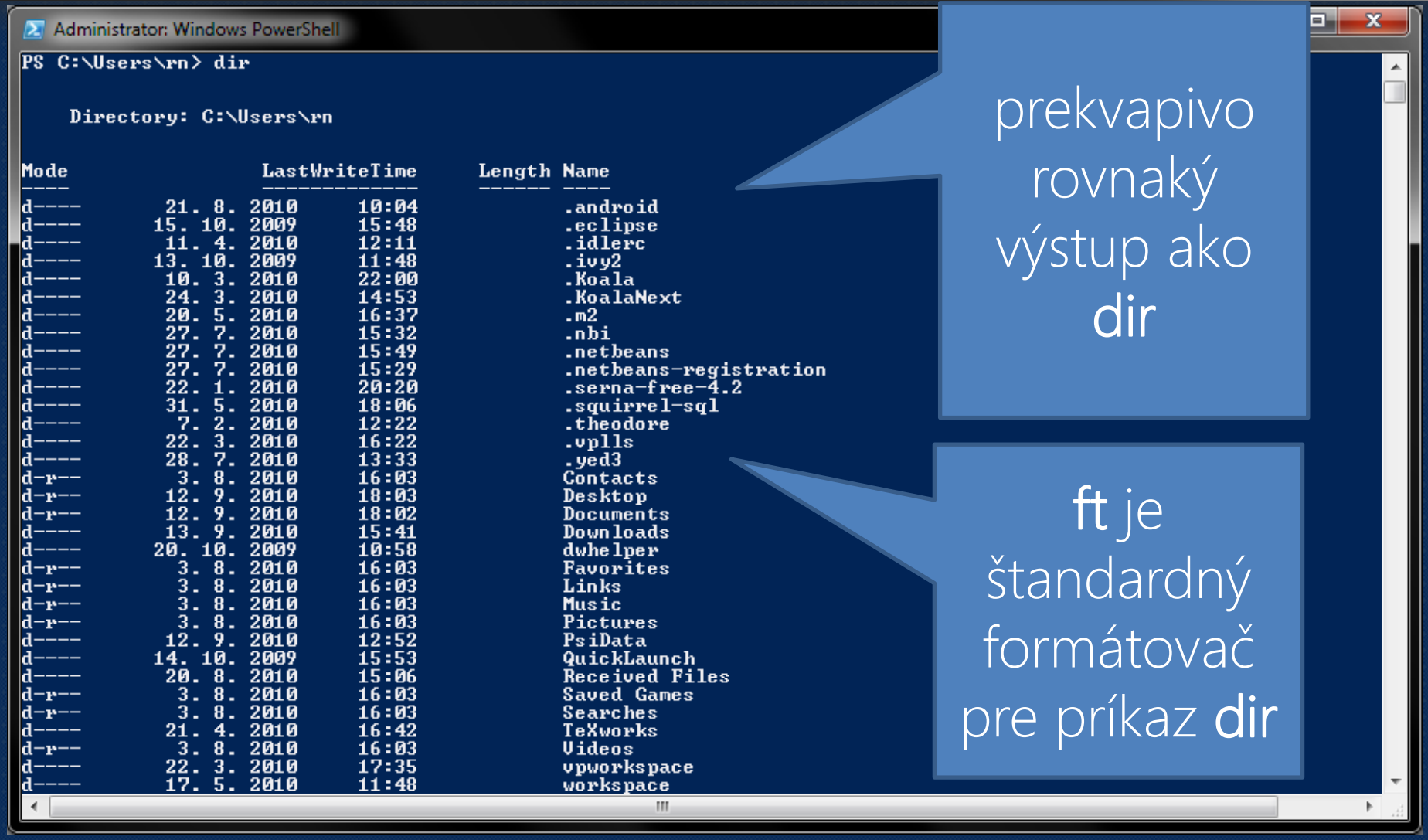
# Iný príklad rúry

```
dir | ft
```

ft = format table

- **dir** vygeneruje zoznam objektov
- **ft** naformátuje zoznam objektov do tabuľky s atribútmi objektov
- znak **|** (pipe, rúra) vytvorí rúru

# Výstup rúry




prekvapivo rovnaký výstup ako **dir**

**ft** je štandardný formátovač pre príkaz **dir**

# „Komplexný“ príklad

Vypíšte názvy a veľkosti všetkých EXE súborov v adresári C:\Windows

```
dir C:\Windows\*.exe | ft name, length
```

- parameter **dir** slúži ako filter názvov súborov
- **ft** zobrazí jednostĺpcovú tabuľku s atribútom **name**
- tabuľka vyplní celú šírku obrazovky

# „Komplexný“ príklad

Vypíšte názvy a veľkosti všetkých EXE súborov v adresári C:\Windows krajším spôsobom

```
dir C:\Windows\*.exe | ft name, length –autosize
```

- **-autosize**: automaticky prispôsobí šírky stĺpcov
- názvy parametrov možno skracovať
  - -a
  - -auto

# Súhrn formátovačov

- Format-Table (ft)
  - naformátuje objekty po riadkoch
  - stĺpce zodpovedajú atribútom
- Format-List (fl)
  - objekty naformátuje pod seba
  - ich atribúty sú tiež pod sebou
- Format-Wide (fw)
  - zobrazí jeden atribút objektov
  - atribúty zobrazí do mriežky

# Trivia time! / Kuriozity

- čo sa stane, keď uvediem len

```
dir
```

- výstup ide do implicitného cmdletu **Out-Default**

```
dir | Out-Default
```

- ten vyberie implicitný formátovač a pošle doň dáta

# Triedenie objektov – Sort-Object

Vypíšte názvy všetkých EXE súborov v adresári C:\Windows usporiadaných podľa veľkosti

- triedenie je univerzálne
- obvykle treba uviesť vlastnosť, podľa ktorej bude triedenie vykonané

trieď podľa dĺžky súboru

```
dir C:\Windows\*.exe | sort -property length
```

# Triedenie objektov

- názov parametra **–property** možno vynechať
- možno triediť aj zostupne

```
dir *.exe | sort name -desc
```

- triedenie štandardne nerozlišuje veľké a malé písmená, možno to zapnúť

```
dir *.exe | sort name –desc -case
```

# Výber niektorých atribútov objektov

- mnohokrát chceme v rúre používať len niektoré z atribútov objektov
- cmdlet **Select-Object** umožňuje odfiltrovať niektoré atribúty
  - alias: **select**

```
dir *.exe | select Name, Length
```

  - vypíše meno a dĺžku súborov

# Rozdiel medzi Select-Object a Format-*

```
dir *.exe | select Name, Length
```

```
dir *.exe | ft Name, Length
```

rovnaký výstup!

- **Select-Object** vezme z rúry objekt a **pošle do rúry** nový objekt, ktorý obsahuje len špecifikované atribúty
- **ft** a **fl** vezme z rúry objekt a **vypíše** len špecifikované atribúty objektu

# Výber prvých N objektov – Select-Object

Vypíšte najväčší súbor v adresári!

```
dir | sort length –desc | select –first 1
```

- **dir** pošle do rúry zoznam súborov/adresárov
- **sort** usporiada objekty podľa dĺžky, najväčší objekt je prvý
- parameter **–first** 1 zoberie zo zoznamu len prvý objekt

Unix: head / tail

# Filtrovanie podľa atribútov

Bežiace služby!

```
Get-Service | Where-Object {$_.Status -eq "Running"}
```

- **Where-Object** vezme z rúry zoznam objektov
- preiteruje tento zoznam
- do rúry pošle len tie objekty, ktoré spĺňajú booleovskú podmienku
- $_: špeciálna premenná, v každej iterácii sa za ňu dosadí aktuálny objekt

# Filtrovanie podľa atribútov

Vypíšte len adresáre v aktuálnom adresári

```
dir | where {$_.PSIsContainer}
```

- atribút **PSIsContainer** je pravdivý, ak je objekt kontajnerom
  - adresár **je** kontajnerom

# Filtrovanie podľa atribútov

Vypíšte len súbory väčšie ako 1MB

```
dir | where {$_.Length -ge 1MB}
```

- **Length**: veľkosť súboru,
- 1MB: výraz rovný 1024
- operátory porovnávania sa nesú v tradičnom duchu shell scriptov...
  - znaky <, > sú rezervované

# Operátory porovnávania

| Operátor | PowerShell |
|---|---|
| > | -gt |
| < | -lt |
| >= | -ge |
| <= | -le |
| = | -eq |
| != | -neq |

existujú aj ďalšie, viď
**help about_Comparison_Operators**

# Skracovanie zápiusov

Vypíšte bežiace služby 2.0

```
gsv | where {$_.Status -eq "Running"}
```

```
gsv | ? {$_.Status -match "run"}
```

- gsv: alias pre Get-Service
- ? a where: aliasy pre Where-Object
- -match: hľadanie podľa regulárneho výrazu

# Štatistické výpočty

- často sú potrebné výpočty
  - sumy
  - počty
  - priemery...
- cmdlet **Measure-Object**
  - alias **measure**

# Štatistické výpočty

Vypíšte počet DOC súborov v aktuálnom adresári

```
dir *.doc | measure
```

- **Measure-Object** vráti objekt, v ktorého atribútoch sú štatistiky
- štandardne sa ráta len počet objektov, ktoré sa nachádzajú v rúre

# Štatistické výpočty

Vypíšte veľkosť, ktorú zaberajú súbory DOC v aktuálnom adresári

```
dir *.doc | measure -sum length
```

- popri počte vráti aj sumu
- ak chceme len sumu, tak:

```
dir *.doc | measure -sum length | select sum
```

# Štatistické výpočty

Koľko cmdletov máme k dispozícii?

```
gcm –CommandType cmdlet | measure
```

```
gcm | ? {$_.CommandType -match "Cmdlet" }
                        | measure
```

# Štatistické výpočty

Koľko riadkov má súbor?

```
Get-Content C:/windows/win.ini | measure -line
```

# DÁTOVÝ MODEL

# Dátový model v PowerShelli

- rúrou putujú objekty
  - v klasickom zmysle OOP
    - properties (inštančné premenné)
    - metódy
- ako zisťovať vlastnosti jednotlivých objektov?

# Dátový model v PowerShelli

- rúrou putujú objekty
  - v klasickom zmysle OOP
    - properties (inštančné premenné)
    - metódy
- ako zisťovať vlastnosti jednotlivých objektov?

cmdlet Get-Member

- získa metadáta o objekte, ktorý príde z rúry

# Informácie o objekte

```
dir | Get-Member
```

```
   TypeName: System.IO.DirectoryInfo

Name                      MemberType     Definition
----                      ----------     ----------
Mode                      CodeProperty   System.String Mode{get=Mode;}
Create                    Method         System.Void Create(System.Security.AccessControl.DirectorySecurity
CreateObjRef              Method         System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
CreateSubdirectory        Method         System.IO.DirectoryInfo CreateSubdirectory(string path), System.IO.
Delete                    Method         System.Void Delete(), System.Void Delete(bool recursive)
Equals                    Method         bool Equals(System.Object obj)
GetAccessControl          Method         System.Security.AccessControl.DirectorySecurity GetAccessControl(),
GetDirectories            Method         System.IO.DirectoryInfo[] GetDirectories(string searchPattern), Sys
GetFiles                  Method         System.IO.FileInfo[] GetFiles(string searchPattern), System.IO.File
GetFileSystemInfos        Method         System.IO.FileSystemInfo[] GetFileSystemInfos(string searchPattern)
GetHashCode               Method         int GetHashCode()
GetLifetimeService        Method         System.Object GetLifetimeService()
GetObjectData             Method         System.Void GetObjectData(System.Runtime.Serialization.Serializatio
GetType                   Method         type GetType()
InitializeLifetimeService Method         System.Object InitializeLifetimeService()
MoveTo                    Method         System.Void MoveTo(string destDirName)
Refresh                   Method         System.Void Refresh()
SetAccessControl          Method         System.Void SetAccessControl(System.Security.AccessControl.Director
ToString                  Method         string ToString()
PSChildName               NoteProperty   System.String PSChildName=.android
PSDrive                   NoteProperty   System.Management.Automation.PSDriveInfo PSDrive=C
PSIsContainer             NoteProperty   System.Boolean PSIsContainer=True
PSParentPath              NoteProperty   System.String PSParentPath=Microsoft.PowerShell.Core\FileSystem::C:
PSPath                    NoteProperty   System.String PSPath=Microsoft.PowerShell.Core\FileSystem::C:\Users
PSProvider                NoteProperty   System.Management.Automation.ProviderInfo PSProvider=Microsoft.Powe
Attributes                Property       System.IO.FileAttributes Attributes {get;set;}
CreationTime              Property       System.DateTime CreationTime {get;set;}
CreationTimeUtc           Property       System.DateTime CreationTimeUtc {get;set;}
```

# Dátový model v PowerShelli

- Objekt v rúre má
  - metódy (methods)
  - vlastnosti/atribúty (properties)

Adresár C: je objekt. Zistite hodnoty jeho vlastností

```
Get-ItemProperty C: | fl *
```

# Get-ItemProperty

```
Get-ItemProperty C: | fl *
```

```
PS C:\Users\rn> Get-ItemProperty C: | fl *


PSPath            : Microsoft.PowerShell.Core\FileSystem::C:\Users\rn
PSParentPath      : Microsoft.PowerShell.Core\FileSystem::C:\Users
PSChildName       : rn
PSDrive           : C
PSProvider        : Microsoft.PowerShell.Core\FileSystem
BaseName          : rn
Mode              : d----
Name              : rn
Parent            : Users
Exists            : True
Root              : C:\
FullName          : C:\Users\rn
Extension         :
CreationTime      : 7. 10. 2009 5:00:21
CreationTimeUtc   : 7. 10. 2009 3:00:21
LastAccessTime    : 26. 9. 2010 9:36:25
LastAccessTimeUtc : 26. 9. 2010 7:36:25
LastWriteTime     : 26. 9. 2010 9:36:25
LastWriteTimeUtc  : 26. 9. 2010 7:36:25
Attributes        : Directory
```

# Dátový model v PowerShelli

Adresár C: je objekt. Zistite jeho dátový typ, metódy a vlastnosti

```
Get-ItemProperty C: | Get-Member
```

```
PS C:\Users\rn> Get-ItemProperty C: | Get-Member


   TypeName: System.IO.DirectoryInfo

Name                      MemberType      Definition
----                      ----------      ----------
Mode                      CodeProperty    System.String Mode{get=Mode;}
Create                    Method          System.Void Create(System.Security.AccessControl.DirectorySecurity d
CreateObjRef              Method          System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
CreateSubdirectory        Method          System.IO.DirectoryInfo CreateSubdirectory(string path), System.IO.D
Delete                    Method          System.Void Delete(), System.Void Delete(bool recursive)
Equals                    Method          bool Equals(System.Object obj)
GetAccessControl          Method          System.Security.AccessControl.DirectorySecurity GetAccessControl(),
GetDirectories            Method          System.IO.DirectoryInfo[] GetDirectories(string searchPattern), Syst
GetFiles                  Method          System.IO.FileInfo[] GetFiles(string searchPattern), System.IO.FileI
GetFileSystemInfos        Method          System.IO.FileSystemInfo[] GetFileSystemInfos(string searchPattern),
GetHashCode               Method          int GetHashCode()
GetLifetimeService        Method          System.Object GetLifetimeService()
GetObjectData             Method          System.Void GetObjectData(System.Runtime.Serialization.Serialization
GetType                   Method          type GetType()
```

# Dátový model: hodnota konkrétnej vlastnosti

Zistite príponu súboru C:\autoexec.bat

```
(Get-Item C:\autoexec.bat).Extension
```

- toto však vieme dosiahnuť aj inak

```
dir C:\autoexec.bat | ft Extension
```

Aký je rozdiel?

# Dátové typy

```
(Get-Item C:\autoexec.bat).Extension
```

- vráti reťazec
- stačí overiť:

```
Get-Item C:\autoexec.bat).Extension | gm
```

```
PS C:\Users\rn> (Get-Item C:\autoexec.bat).Extension | gm

   TypeName: System.String

Name                 MemberType           Definition
----                 ----------           ----------
Clone                Method               System.Object Clone()
CompareTo            Method               int CompareTo(System.Object value), in
```

# Dátové typy

```
dir C:\autoexec.bat | ft Extension
```

- vracia formátované objekty

```
PS C:\Users\rn> dir C:\autoexec.bat | ft Extension | gm

   TypeName: Microsoft.PowerShell.Commands.Internal.Format.FormatStartData

Name                                       MemberType Definition
----                                       ---------- ----------
Equals                                     Method     bool Equals(System.Object obj)
GetHashCode                                Method     int GetHashCode()
GetType                                    Method     type GetType()
ToString                                   Method     string ToString()
autosizeInfo                               Property   Microsoft.PowerShell.Commands.Inter
ClassId2e4f51ef21dd47e99d3c952918aff9cd    Property   System.String ClassId2e4f51ef21dd47
groupingEntry                              Property   Microsoft.PowerShell.Commands.Inter
pageFooterEntry                            Property   Microsoft.PowerShell.Commands.Inter
pageHeaderEntry                            Property   Microsoft.PowerShell.Commands.Inter
shapeInfo                                  Property   Microsoft.PowerShell.Commands.Inter

   TypeName: Microsoft.PowerShell.Commands.Internal.Format.GroupStartData
```

# Dátové typy

- vidíme, že PowerShell má silný objektový model
- podporované sú všetky objekty .NET frameworku
- objekty sú obalené do špeciálneho objektu PSObject, ktorý rozširuje funkcionalitu o niektoré pomocné metódy

# Dátové typy

Zistite hodnotu pi

```
[System.Math]::Pi
```

- **Math** je trieda z menného priestoru System v .NET
- :: reprezentuje volanie statickej premennej **Pi**
- triedy zo System majú skrátené názvy:

```
[Math]::Pi
```

# Dátové typy - metódy

- metódy voláme rovnako ako vlastnosti

```
(Get-Item D:\notepad.txt).Delete()
```

```
[URI]::EscapeUriString("http://upjs.sk/ĽudoŠtúr")
```

# Dátové typy – vytváranie inštancií

- v PowerShelli možno vytvoriť inštanciu ľubovoľnej .NET triedy a veselo ju používať

cmdlet New-Object

Stiahnite RSSko zo SME.sk

```
(New-Object System.Net.WebClient).DownloadString("http://www.google.com")
```

# PROVIDERS

# Zjednotenie prístupu k dátovým zdrojom

- filozofia prístupu k súborovému systému je dôverne známa
  - diskové jednotky, adresáre, súbory
  - pohybujeme sa v nich cez **cd**, vypisujeme cez **dir**
- PowerShell rozširuje túto filozofiu na rozličné dátové zdroje
  - registre
  - SQL server
  - ActiveDirectory...

# Zjednotenie prístupu k dátovým zdrojom

Get-PSProvider

- vypíše zoznam providerov
- každý provider poskytuje prístup k „jednotkám“

Get-PSDrive

- PSDrive pre súborový systém = C:\, D:\
- PSDrive pre registre = HKLM:, HKCU:
  PSDrive pre premenné prostredia = ENV:\

# Registre ako dátový zdroj

Zistite cestu k obrázku na pracovnej ploche

- nachádza sa v registroch HKEY_CURRENT_USER\Control Panel\Desktop, v kľúči **Wallpaper**

```
Get-Item "HKCU:\Control Panel\Desktop\" | gm
```

- položka je typu Microsoft.Win32.RegistryKey
- požadovaná sa nachádza vo vlastnosti Wallpaper

# Registre ako dátový zdroj

Zistite cestu k obrázku na pracovnej ploche

```
(Get-Item "HKCU:\Control Panel\Desktop")
  .GetValue("WallPaper")
```

```
Get-ItemProperty
  -Path "HKCU:\Control Panel\Desktop"
  -name Wallpaper | ft Wallpaper
```

# Registre ako dátový zdroj

- vieme využívať štandardné príkazy
  - cd (presun po štruktúre)
  - mkdir (vytvára kľúče)
- vieme nastavovať hodnoty

```
Set-ItemProperty
  -path HKLM:\Software\MyCompany
  -name NoOfEmployees -value 824
```

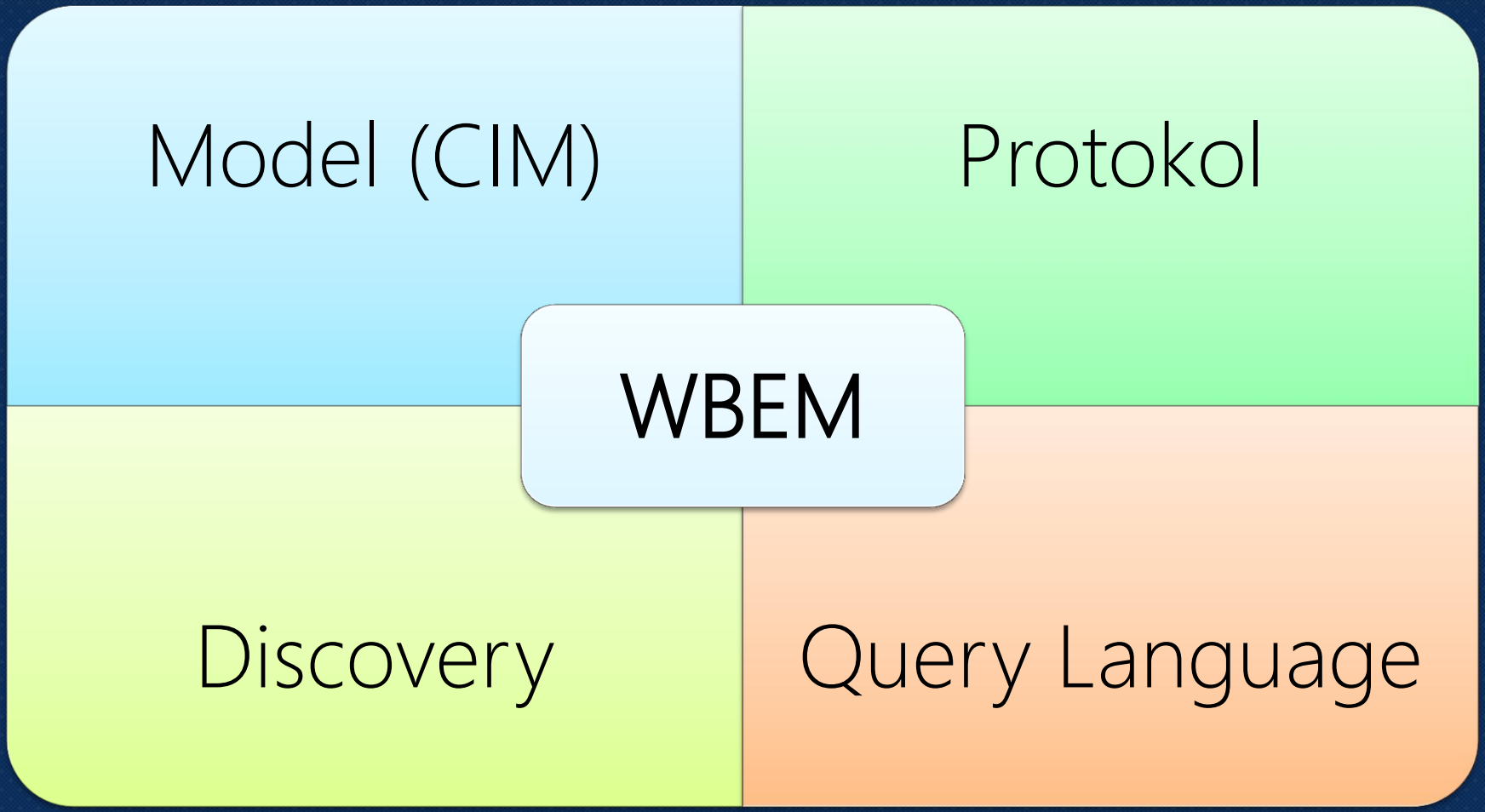
# Monitorovanie a správa zariadení v sieti?

- už dávno-pradávno vznikol protokol SMNP (Simple Network Management Protocol)
  - na aplikačnej vrstve ISO-OSI modelu
- problémy:
  - návrh z konca 80. rokov
  - zariadenia s malým výkonom = protokol musel byť čo najjednoduchší = limitovaná funkčnosť
  - výrobcovia časom začali robiť vlastné rozšírenia protokolu

# Web-Based Enterprise Management


- infraštrukatúra pre správu komponentov systému = WBEM
- implementácia WBEM v rozličných systémoch
  - Microsoft: Windows Management Instrumentation (WMI)
    - od verzie Wnidows 98
  - Apple Remote Desktop, SUSE Linux Enterprise Server, Solaris WBEM Services, Red Had Enterprise Linux, Ubuntu...

# Zložky WBEM



# Windows Management Instrumentation

- Common Information Model (CIM)
- model manažovateľných entít
- entitou môže byť čokoľvek: hardvérový komponent, softvér, služba, ale i celé zariadenie
- OOP filozofia
  - objekt = vlastnosti (properties) + operácie

# Windows Management Instrumentation

- transportný protokol
  - protokol pre komunikáciu s objektmi
  - Microsoft zvolil DCOM
  - od Visty alternatíva: Windows Remote Management (Win-RM)
    - interoperabilná technológia webových služieb: SOAP cez HTTP

# Object Discovery

- ako vyhľadávať entity?
- ako ich adresovať?
  - ako adresovať tretiu inštanciu mspaint.exe na piatom serveri?
- adresácia pomocou ciest:

```
\\alef0\root\cimv2:Win32_Process.Handle="321"
```

# Dopytovanie

- štandardom CIM Query Language
  - podmnožina SQL: len SELECTy
- Microsoft opäť išiel pod svojom: WQL
  - WMI Query Language
  - rovnaká filozofia
- objekty sa vnímajú ako tabuľky

```
SELECT Name FROM Win32_Process
  WHERE Name Like "notepad"
```

# WMI a PowerShell

- podpora priamo zabudovaná
- cmdlet **Get-WmiObject**

Vypíšte zoznam všetkých bežiacich procesov

```
Get-WmiObject Win32_Process | ft Caption,
                               CommandLine
```

# Get-WmiObject

Vypíšte zoznam všetkých bežiacich procesov spustených z C:\Windows

```
Get-WmiObject Win32_Process
-filter "ExecutablePath LIKE 'C:\\Windows%'"
| ft Name, ExecutablePath
```

filter je v jazyku WQL!

# Get-WmiObject

- niektoré činnosti podporuje priamo PowerShell
- WMI umožňuje získavať naozaj hĺbkové informácie

Zistite verziu BIOSu v aktuálnom počítači

```
Get-WmiObject Win32_BIOS | ft SMBIOSBIOSVersion
```

# Odkiaľ zistím, čo sa dá zistiť z WMI?

- Dokumentácia na MSDN udáva zoznam všetkých tried, ich vlastností a metód:
  - http://msdn.microsoft.com/en-us/library/aa394084%28v=VS.85%29.aspx

# Uľahčenie zápisu

- niektoré činnosti možno zapísať skrátene:

Zistite informácie o službe dnscache

pretypovanie reťazca na objekt typu wmi

```
[wmi] 'Win32_Service.Name="Dnscache"'
```

# Skriptovacie finty

- PowerShell umožňuje mnohoraké finty uľahčujúce skriptovanie
- hlavne:
  - aliasy
  - skracovanie názvov parametrov

# Názvy cmdletov


- štandardný názov cmdletu:

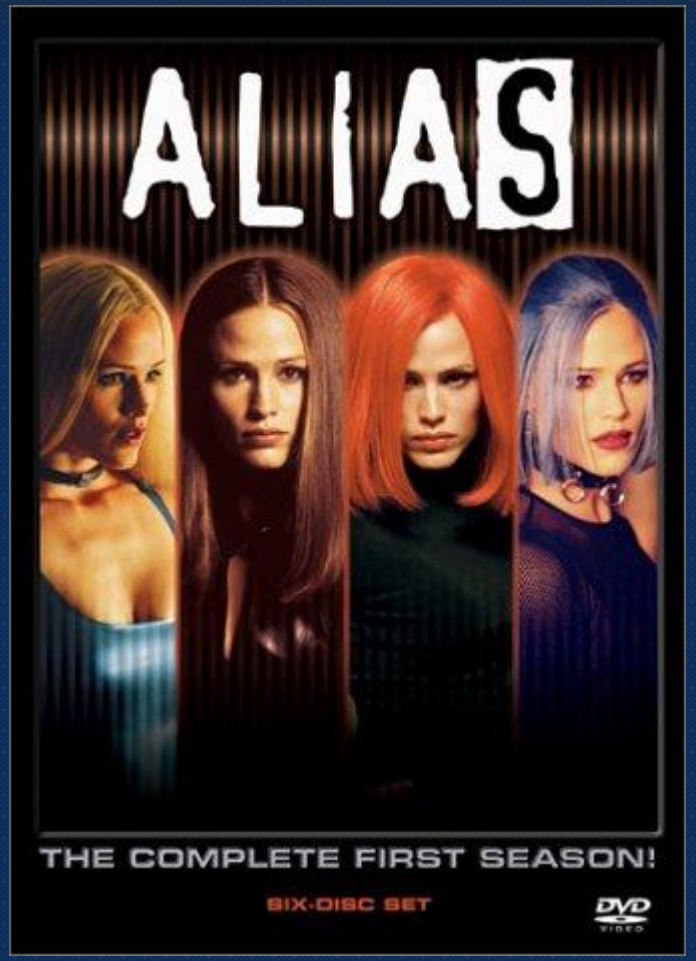
*sloveso-vec*
*prísudok-predmet*

- Get-ChildItem
- Set-Date
- Copy-Item

```
Get-Command
```

# Aliasy

- cmdlet možno volať pod rozličnými názvami
- aliasy šetria písanie
- uľahčujú kultúrny šok pri prechode z iných systémov
- zneprehľadňujú kód



# Aliasy

| Definícia | Alias |
| --- | --- |
| Get-ChildItem | ls, dir, gci |
| Get-Command | gcm |
| Copy-Item | cp, copy, cpi |
| Where-Object | where, ? |
| ForEach-Object | foreach, % |
| Get-Command | gcm |
| Get-Alias | alias, gal |

# Skracovanie názvov parametrov

- názvy parametrov možno skracovať dovtedy, pokiaľ sú jednoznačné
- v niektorých prípadoch môžeme vynechať meno pomenovaného parametra

```
Get-ChildItem -Path D:/MP3 -Filter O* -Recursive
```

```
dir d:/MP3 o* -r
```

# Problém: zneprehľadňovanie

- Čo robí tento one-liner?

```
dir d:/mp3 | ? {$_.PSIsContainer}
 | select FullName,
          { (dir $_.FullName -r
             | measure -s length).Sum }
```

# PowerShell a viacriadkové skripty

- PowerShell podporuje plnoprávne programovanie
- vlastná syntax + arzenál .NET frameworku
  - objekty
  - funkcie
  - ...
- dodáva sa PowerShell ISE (Integrated Script Environment)

# Vypíšte veľkosti adresárov

```
function Velkost($adresar) {
    $meno = $adresar.FullName
    $statistiky = Get-ChildItem $meno -Recurse
        | Measure-Object -Property Length -Sum
    $statistiky.sum
}

$adresare = Get-ChildItem
              | Where {$_.PSIsContainer}

foreach($adresar in $adresare) {
    $velkost = Velkost($adresar)
    Write-Host $adresar.FullName $velkost
}
```

# Funkcie sa tvária ako cmdlety

- pozor na neobvyklú syntax volania

```
function Get-Sum($a, $b) {
    return $a + $b
}

Get-Sum(3, 5)
```

```
3
5
```

```
Get-Sum 3 5
```

# Parametre funkcií

```
function Get-Sequence([int] $from, [int] $to,
                      [int] $step = 1)
{
    for($i = $from; $i -le $to; $i = $i + $step) {
        $i
    }
}
```

```
Get-Sequence 0 20 -step 2
```

# Funkcie, ktoré spracovávajú pipeliny

- špeciálna premenná **$input** s objektami, ktoré prídu z rúry

```
function Measure-Directory {
  foreach($item in $input) {
    $size = (dir $item.FullName -Recurse
      | Measure-Object -Sum -Property Length).Sum

    @{$item.FullName = $size}
  }
}
```

```
dir D:\MP3 | Measure-Directory
```

# Príklad: očíslujte položky z rúry

```
function Enumerate-Items {
    $itemIndex = 0;

    foreach($item in $input) {
        @{$itemIndex = $item}
        $itemIndex++
    }
}

dir | Enumerate-Items
```

# Príklad: očíslujte položky z rúry II.

- alternatívna syntax funkcií nad rúrou
  - **begin**: vykoná sa na začiatku rúry
  - **process**: pre každú položku v rúre.
    - aktuálna položka je v premennej $_
  - **end**: na konci rúry

```
function Enumerate-Items {
    begin {
        $itemIndex = 0;
    }

    process {
        @{$itemIndex = $_}
        $itemIndex++
    }
}
```

# Príklad: očíslujte položky z rúry III.

- alternatívna alternatívna syntax funkcií nad rúrou

```
filter Get-CpuHeavyProcess {
  if($_.PercentProcessorTime -ge 2) {
    $obj = New-Object PSObject | Select Name, CPU
    $obj.Name = $_.Name;
    $obj.CPU = $_.PercentProcessorTime;

    $obj
  }
}

gwmi Win32_PerfFormattedData_PerfProc_Process
  | Get-CpuHeavyProcess
```

# Príklad: očíslujte položky z rúry III.

- filter je sémanticky ekvivalentný funkcii s jediným blokom **process**

```
filter Get-FilteredObject
{
   ...
}
```

```
function Get-FilteredObject
{
   process {
   }
}
```

Aktuálna položka: $_

Aktuálna položka: $input.current

# Vývoj cmdletov v C#

- vývoj sa ničím nelíši od vývoja iných tried
- vytvoríme novú **Class Library** (.DLL)
- základná trieda cmdletu dedí od PSCmdlet
  - System.Management.Automation.PSCmdlet
- prekryjeme metódu **ProcessRecord()**
  - protected override void ProcessRecord()
- metódou **WriteObject()** zapisujeme do rúry

# Vývoj cmdletov v C#

```
public class GetHelloCmdlet : PSCmdlet {
  protected override void ProcessRecord()
{
     WriteObject("Hello World!");
  }
}
```

- trieda musí mať atribút Cmdlet
  - System.Management.Automation.CmdletAttribute

# Vývoj cmdletov v C#

- pred nainštalovaním potrebujeme ešte triedu pre **snap-in** (zásuvný modul)
- obsahuje metadáta o inštalovanom cmdlete
- potrebujeme vlastnú triedu dediacu od **PSSnapIn**
- prekryť properties:
  - Name: názov snap-inu
  - Vendor: dodávateľ
  - Description: popis

# Vývoj cmdletov v C#

```
[RunInstaller(true)]
public class HelloCmdlets : PSSnapIn {
        public override string Description {
            get { return "Powershell Greetings"; }
        }

        public override string Vendor {
            get { return "UPJS"; }
        }

        public override string Name {
            get { return "HelloCmdlets"; }
        }
}
```

# Inštalácia cmdletu

- prekompilovať projekt
- z PowerShellu vojsť do adresára s DLL
- importnúť modul

```
Import-Module .\HelloWorldCmdlets.dll
```

- overiť nainštalovanie cez

```
Get-Module
```

# Alternatívna inštalácia cmdletu

- prekompilovať projekt
- použitím installutil nainštalovať do systému
  - installutil.exe v adresári .NET frameworku
- v PowerShelli zistiť zoznam registrovaných snap-inov
- pridať snap-in do aktuálneho sessionu

```
installutil [názovknižnice].dll
```

```
Get-PSSnapin -Registered
```

```
Add-PSSnapin HelloCmdlets
```

# Parametre

- cmdlet môže mať vlastné parametre
- deklarujeme ich ako **klasické properties** v triede
- nezabudnúť na atribút [**Parameter**]!

```
[Parameter]
[ValidateSet("frontpage", "smenajcit4")]
public String Channel { get; set; }
```

parameter s pevne danými možnosťami

# Pozičné parametre

- nemusíme udávať názov parametra

prvý parameter zľava je **Word**

```
[Parameter(Position=0) ]
public String Word { get; set; }
```

Get-Translation [[-Word] <String>]

```
Get-Translation "hello"
```

# Povinné parametre

- nastaviť ako parameter atribútu Parameter ;-)

povinný parameter

```
[Parameter(Mandatory=true) ]
public String Word { get; set; }
```

Get-Translation -Word <String>

# Hodnoty parametrov z rúry

- hodnota parametra môže byť prevzatá z rúry

```
[Parameter(ValueFromPipeline=true, Position=0)]
public String Word { get; set; }
```

```
"Hello", "World" | Get-Translation
```

- pred každým zavolaním **ProcessRecord()** sa objekt z rúry namapuje na príslušný parameter
  - v ukážke: objekt z rúry sa namapuje na parameter Word
  - prebehne automatická konverzia typov

# Hodnoty parametrov z rúry

- voliteľne môžeme na parameter namapovať konkrétnu property objektu z rúry
- príklad: na parameter Word sa namapuje vlastnosť Word z objektu v rúre

```
[Parameter(ValueFromPipelineByPropertyName = true,
           ValueFromPipeline=true, Position=0)]
public String Word { get; set; }
```

- ValueFromPipeLine má prednosť pred Value..ByPropertyName

# Hlásenie chýb

- WriteError()
  - pre chyby, ktoré nebránia v spracovávaní ďalších objektov v rúre
- ThrowTerminatingError()
  - kritické chyby, spracovanie rúry sa zastaví
- metódy zdedené z PSCmdlet-u

http://msdn.microsoft.com/en-us/library/ms714414(VS.85).aspx

# Hlásenie chýb: WriteError

```
if (!slovnik.ContainsKey(Word)) {
  WriteError(
     new ErrorRecord(
        new KeyNotFoundException(),
          "No such word in the dictionary.",
          ErrorCategory.InvalidArgument,
          Word));
}
```

- WriteError: vyžaduje inštanciu ErrorRecord
- ErrorRecord: 4 parametre:
  - výnimka, popis, kategória chyby a objekt, ktorý sa spracovával

```
Get-Translation : The given key was not present in the dictionary
At line:1 char:41
+ "hello", "ah", "world" | Get-Translation <<<<
    + CategoryInfo          : InvalidArgument: (ah:String) [Get-T
    + FullyQualifiedErrorId : No such word in the dictionary.,UP
```

# Hlásenie chýb: ThrowTerminatingError

- rovnaká syntax ako v prípade WriteError
- PowerShell vyhodí výnimku **PipelineStoppedException**
- ďalšie objekty sa už nebudú spracovávať

# Tutoriál na webe k vytváraniu cmdletov

http://ics.upjs.sk/~novotnyr/wiki/PowerShell/VytvaranieCmdletovVSharpDevelop