

# Powershell

Kristýna Petrlíková

Školní rok 2019/2020

- 1 Úvod
- 2 Syntax
- 3 Pokročilé

- 1 Úvod
  - Obecné informace
  - Desktop vs. Core
  - Instalace
  - Vlastnosti jazyka
- 2 Syntax
- 3 Pokročilé

- Textové prostředí (*shell*) se skriptovacím jazykem
- Vyvíjené společností Microsoft od roku 2006
  - Podrobná oficiální dokumentace
- Založeno na platformě .NET (předtím Framework, nyní Core)
- V posledních letech se rozšiřuje na ostatní systémy
  - Linux, Mac OS, ARM (Raspberry Pi)

# Windows Powershell

- Poslední verze 5.1
- Založený na .NET Framework
- Dostupný pouze na Windows
- Na systému již předinstalovaný
- Vývoj byl ukončen - nyní se pouze udržuje
  - Stále se vydávají opravy chyb a bezpečnostní záplaty



- Uveden v srpnu 2016 s verzí 6.0
- Založený na .NET Core Runtime
- Poslední verze 7.0.0 (experimentální)
- Open-source
  - Instalace z balíčku na [Githubu](#) (zde je popsán podrobný návod pro všechny systémy)



## Get PowerShell

You can download and install a PowerShell package for any of the following platforms.

Supported Platform	Download (LTS)	Downloads (stable)	Downloads (preview)	How to Install
<a href="#">Windows (x64)</a>	<a href="#">.msi</a>	<a href="#">.msi</a>	<a href="#">.msi</a>	<a href="#">Instructions</a>
<a href="#">Windows (x86)</a>	<a href="#">.msi</a>	<a href="#">.msi</a>	<a href="#">.msi</a>	<a href="#">Instructions</a>
<a href="#">Ubuntu 18.04</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">Instructions</a>
<a href="#">Ubuntu 16.04</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">Instructions</a>
<a href="#">Debian 9</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">Instructions</a>
<a href="#">Debian 10</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	<a href="#">.deb</a>	
<a href="#">CentOS 7</a>	<a href="#">.rpm</a>	<a href="#">.rpm</a>	<a href="#">.rpm</a>	<a href="#">Instructions</a>

- Multiplatformní
- Velká část příkazů z verze 5.1 kvůli kompatibilitě zatím chybí

# Instalace na Windows

Z **Githubového repozitáře** Powershellu (sekce Assets v Releases nebo Get Powershell v README) stáhneme instalační program pro náš systém.

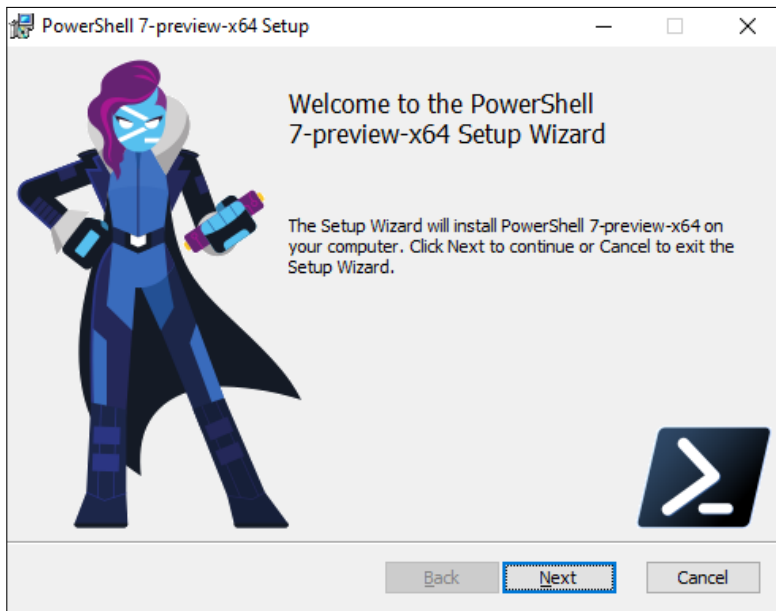
 PowerShell-7.1.0-preview.2-win-x64.msi

86.9 MB

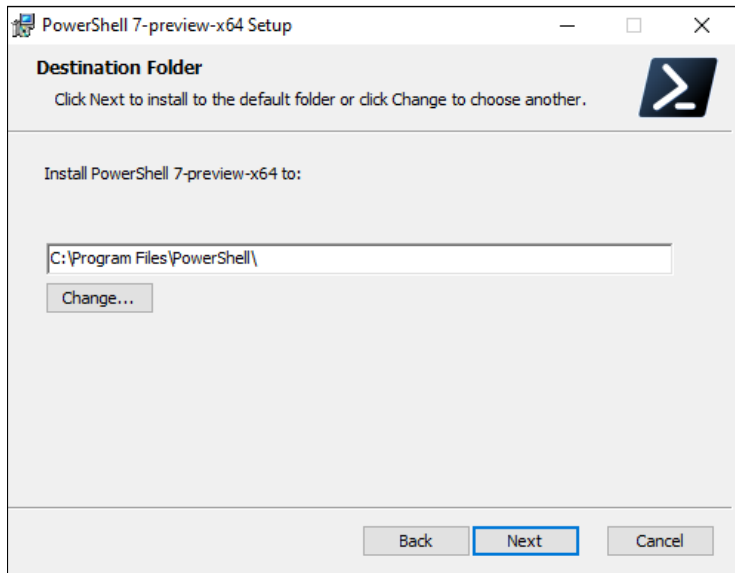
Při instalaci klikáme na tlačítko Next, dokud se neobjeví tlačítko Install, na které také klikneme.



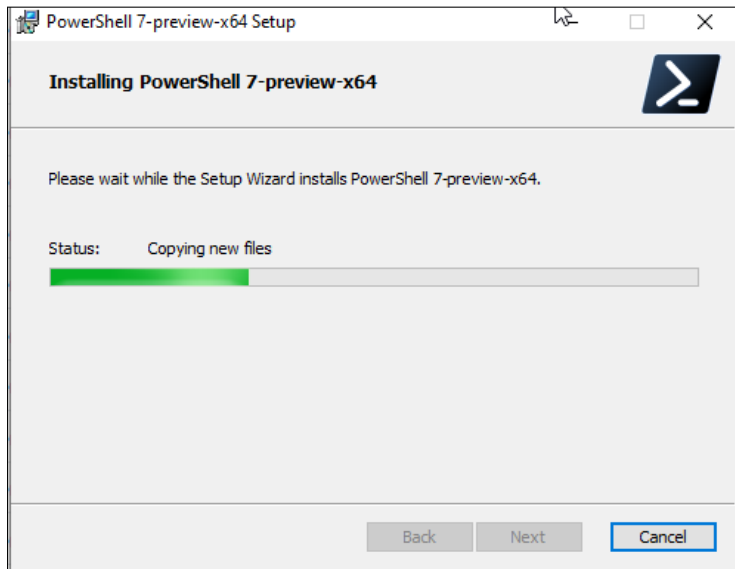
# Instalace na Windows



# Instalace na Windows

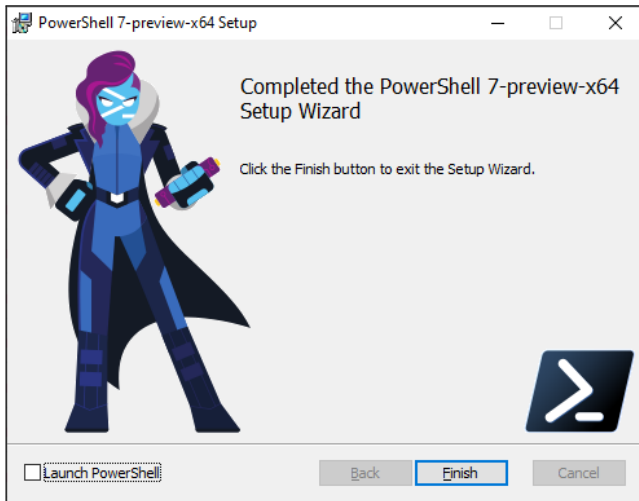


# Instalace na Windows

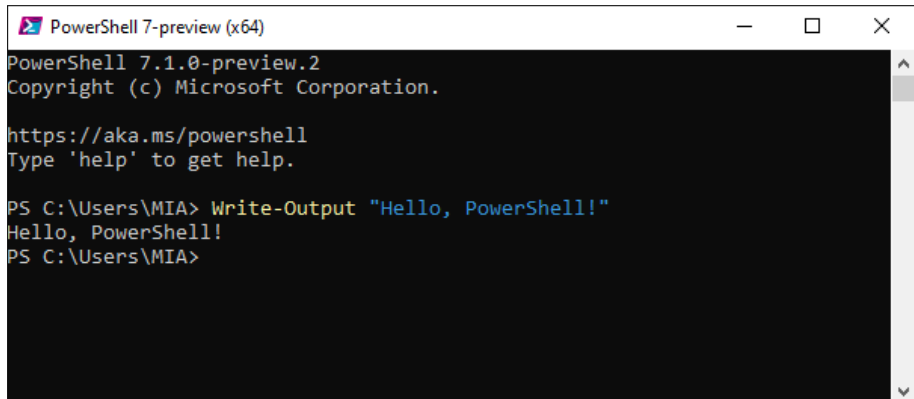


# Instalace na Windows

Klikneme na tlačítko Finish.



# Instalace na Windows



The screenshot shows a PowerShell 7-preview (x64) window. The title bar includes the PowerShell icon and the text "PowerShell 7-preview (x64)". The window has standard Windows window controls (minimize, maximize, close). The console output is as follows:

```
PowerShell 7.1.0-preview.2  
Copyright (c) Microsoft Corporation.  
  
https://aka.ms/powershell  
Type 'help' to get help.  
  
PS C:\Users\MIA> Write-Output "Hello, PowerShell!"  
Hello, PowerShell!  
PS C:\Users\MIA>
```

# Instalace na Arch Linux

- Neoficiální podpora ze strany uživatelské komunity
- Dostupné jako balíček `powershell-bin` v Arch User Repository
- Instaluje se pomocí správce balíčků, který podporuje AUR

# Instalace na Arch Linux

```
[blanche@arch ~]$ yay -Sy powershell-bin
```

```
[sudo] password for blanche:
```

```
(1/4) installing liburcu
```

```
(2/4) installing numactl
```

```
(3/4) installing lttng-ust
```

```
(4/4) installing openssl-1.0
```

```
Total Installed Size: 151.00 MiB
```

```
:: Proceed with installation? [Y/n]
```

```
(1/1) Arming ConditionNeedsUpdate...
```

# Instalace na Arch Linux

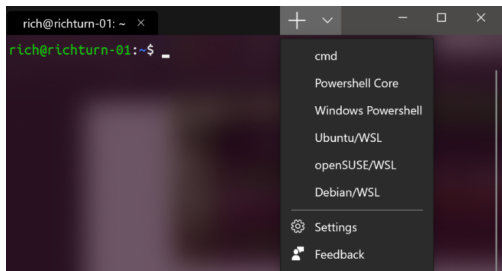
```
[blanche@arch ~]$ pwsh
PowerShell 7.0.0
Copyright (c) Microsoft Corporation. All rights reserved.

https://aka.ms/powershell
PS /home/blanche> Write-Output "Hello, PowerShell!"
Hello, PowerShell!
```



# Windows Terminal

- Představen v květnu 2019
- Dostupný ve Windows Store
- Open-source
- Moderní design konzole pro Windows 10
- Umožňuje používat více panelů
  - V každém z nich lze spustit klasický CMD, PowerShell, nebo třeba linuxový terminál (možnost rozšíření)
- Podpora ligatury (slitek) kódu



- Přípona `.ps1`
- Objektově orientovaný se vším všudy
  - Výstupy příkazů jsou (ve většině případů) také objekty
  - Tím se liší například od klasického unixového shellu, kde jsou příkazy schopny vrátit pouze text
- Příkazy mohou, ale nemusí být zakončeny středníky (v případě, že chceme dostat více příkazů na jeden řádek)
  - Stačí je oddělovat koncem řádku (*newline*)
- Možnost spouštět skripty samostatně nebo psát příkazy do interaktivní konzole
- Načítání modulů (externí balíčky příkazů) - soubory `.psm1` nebo `.dll`

- **Pipeline** (roury) - přesměrování vstupu mezi příkazy
- **Cmdlet** - příkaz ve formátu Sloveso-Podstatné jméno
  - např. `Get-Command`, `Write-Output`, `Add-Content`
- Case-insensitive (nezáleží na velikosti písmen)
  - `Get-Help` znamená to samé jako `get-help`
- Podpora **aliasů**
- Možnost kombinace s unixovými příkazy (na které buď existuje alias pro cmdlet - `cat`, `pwd`, `echo`, nebo jsou přímou součástí systému - `lolcat`, `cmatrix`, `cowsay`)
  - Nedoporučuje se, skript pak nemusí být spustitelný na jiných systémech

## 1 Úvod

## 2 Syntax

- Proměnné
- Cmdlety
- Operátory
- Větvení
- Cykly
- Pipeline
- Funkce
- Objekty

## 3 Pokročilé

# Proměnné

- Slouží k ukládání informací do paměti
- Zapisují se s \$ na začátku
- Na velikosti písmen při zápisu nezáleží (*Case-insensitive*)
- K přiřazení hodnoty se používá = (*inicializace*)
- Hodnota se ale nemusí přiřadit, pokud proměnnou pouze nazveme (*deklarace*)
- Typ proměnné rovněž nemusí být upřesněn, ale dá se zapsat v [ ] před její název
  - Případně se tak proměnná dá přetypovat

## Typovaná deklarace

```
[string]$greetings
```

## Netypovaná inicializace

```
$greetings = "Ahoj světe!"
```

# Přehled běžně používaných typů

- Přejaté z platformy .NET
- K zjištění typu proměnné se používá zřetězení metody a členu `.GetType().FullName`

Název	Popis	Příklad
[char]	Jeden znak (s podporou Unicode)	'😊'
[string]	Textový řetězec - posloupnost znaků	"red"
[bool]	Logická hodnota	\$true, \$false
[int]	Celé číslo (32-bit)	42
[long]	Celé číslo (64-bit)	1982989219753
[float]	Desetinné číslo (přesnost ~7 řádů)	3.141593
[double]	Desetinné číslo (přesnost ~15 řádů)	3.14159265359
[decimal]	Desetinné číslo (přesnost ~28 řádů)	0.1234567891011
[DateTime]	Datum a čas	Get-Date
[Object[]]	Kolekce prvků	@('a', "B", 2.7)

# Komentáře

- Kód, který není vyhodnocován jako příkaz
- Používá se k dokumentaci kódu nebo (dočasněmu) odebrání některé funkcionality
- Jednořádkový (tj. do konce řádku) začíná s #
- Víceřádkový je pak obsažen v <# #>

## Jeden řádek

```
# Pozdraví svět
```

```
Write-Output "Ahoj světe!"
```

## Více řádků

```
<#
```

```
Toto je velice důležitá proměnná.  
Už jsem sice zapomněl, co dělá,  
ale pokud se odstraní,  
program začne náhodně padat.
```

```
#>
```

```
$x = 42
```

- Kompilovaný kód C# (nebo čehokoliv, co podporuje .NET)
- Zkratka pro "Command let"
- Příkaz ve formátu Sloveso-PodstatnéJméno
  - *Sloveso* vyjadřuje prováděnou operaci
  - *Podstatným jménem* (standardně v jednotném čísle) označíme objekty, na kterých operaci vykonáme
- Kromě tohoto pravidla by se také měla používat standardní slovesa
  - Dají se zobrazit příkazem Get-Verb
- Na část z nich se lze odkázat přes aliasy
  - Definované zvlášť pro slovesa a podstatná jména, kombinují se dohromady



## Výběr některých běžně používaných sloves v Cmdletech

Název	Alias	Příklad	Alias příkladu
Add	a	Add-Content	
Clear	cl	Clear-History	clhy
Copy	cp	Copy-Item	cpi
Get	g	Get-Process	gps
Find	fd	Find-Module	
Format	f	Format-Table	ft
Join	j	Join-Path	
Move	m	Move-Item	mi
New	n	New-Alias	nal
Remove	r	Remove-Variable	rv
Set	s	Set-Location	sl
Show	sh	Show-Markdown	
Split	sl	Split-Path	

## Sčítání

- Podporováno mezi čísly, řetězci, poli a slovníky

Výraz	Výsledek
<code>41 + 1</code>	<code>42</code>
<code>"abc" + "def"</code>	<code>"abcdef"</code>
<code>@(1, "jedna") + @(2.0, "dva")</code>	<code>@(1, "jedna", 2.0, "dva")</code>
<code>@{'A' = "Alfa"} + @{'B' = "Bravo"}</code>	<code>@{'A' = "Alfa"; 'B' = "Bravo"}</code>

## Odčítání

- Možné pouze u čísel
- Binární (mezi dvěma čísly):  $18 - 8$
- Unární (převrácení hodnoty):  $-7$

## Násobení

- Definováno v rámci čísel a polí

Výraz	Výsledek
$6 * 7$	42
$"abc" * 5$	<b>"abcabcabcabcabc"</b>
$@(7, 1) * 3$	$@(7, 1, 7, 1, 7, 1)$

## Dělení

- Pokud není určený typ, výsledek se dle potřeby dokáže převést buď na celé, nebo desetinné číslo
- `(6/3).GetType().FullName` → `System.Int32` (název v .NET)
- `(6/4).GetType().FullName` → `System.Double`

## Modulo

- Vrací zbytek po dělení dvou (klidně i desetinných) čísel
- `16 % 5` → `1`
- `5.3 % 2` → `1.3`

## Ostatní matematické funkce

- Dostupné v .NET třídě `Math`
- `[Math]::floor([Math]::pi)` # 3

# Aritmetika

## Aritmetika v rámci bitů (bitwise arithmetic)

- Pracuje s každým bitem čísla zvlášť
- Číslo lze zapsat v binární formě s prefixem 0b, například 0b111 → 7

Název operace	Operátor	Výsledek na bitech $a(, b)$	Příklad
Binární součin	-band	$a \cdot b$	$\begin{array}{r} 0b1100 \\ \text{AND } 0b1010 \\ \hline 0b1000 \end{array}$
Binární součet	-bor	$a + b$	$\begin{array}{r} 0b1100 \\ \text{OR } 0b1010 \\ \hline 0b1110 \end{array}$
Exkluzivní součet	-bxor	$a \neq b$	$\begin{array}{r} 0b1100 \\ \text{XOR } 0b1010 \\ \hline 0b0110 \end{array}$
Binární negace	-bnot	$\neg a$	$\begin{array}{r l} a & 0b10101110 \\ \hline \neg a & 0b01010001 \end{array}$
Posun vlevo	-shl		$\begin{array}{r l} a & 0b10101110 \\ \hline a \ll 3 & 0b01110000 \end{array}$
Posun vpravo	-shr		$\begin{array}{r l} a & 0b10101110 \\ \hline a \gg 3 & 0b00010101 \end{array}$

# Porovnávání

Název operace	Operátor	Pravdivý výraz	Nepravdivý výraz
Rovná se	-eq	"abc" -eq "ABC"	5 -eq 8
Nerovná se	-ne	"abc" -ne "def"	42 -ne 42
Větší než	-gt	9 -gt 5	5 -gt 5
Větší nebo rovno	-ge	8 -ge 8	7 -ge 12
Menší než	-lt	12 -lt 24	24 -lt 12
Menší nebo rovno	-le	'a' -le "z"	'f' -le 'd'
Wildcard match	-like	"ABCDEF" -like "a*"	"ABCDEF" -like "b*"
Regex match	-match	"ABCDEF" -match "BC"	"ABCDEF" -match "^B*"
Přítomnost	-in	"a" -in @('a','b','c')	"a" -in @("b","c")
Typová rovnost	-is	'a' -is [string]	5 -is [char]

Větvení programu na základě pravdivosti logických výrazů.

## Příkaz If/Else

- Skládá se z částí *If*, *Elseif* a *Else*
- *Elseif* a *Else* jsou nepovinné (ale jejich pořadí se musí dodržovat)
- *Elseif* může být zopakováno neomezeně
- Jakmile se program do jedné z větví zanoří, po vykonání příkazů skočí za konec *Else* (nebo za poslední část v sérii)

## Obecný zápis

```
if( <podmínka1> ) {  
    <příkazy>  
}  
elseif( <podmínka2> ) {  
    <příkazy>  
}  
# ...  
elseif( <podmínkaN> ) {  
    <příkazy>  
}  
else {  
    <příkazy>  
}
```



## Příklad: BMI kalkulačka

```
$weight = Read-Host "Zadejte hmotnost (v kg)"
$height = (Read-Host "Zadejte výšku (v cm)") / 100

$bmi = $weight / ($height * $height)

Write-Host "BMI:" $bmi

if ($bmi -le 16.5) {
    Write-Host -ForegroundColor Blue "Těžká podvýživa."
}
elseif ($bmi -le 18.5) {
    Write-Host -ForegroundColor Cyan "Podváha."
}
elseif ($bmi -le 25) {
    Write-Host -ForegroundColor Green "Ideální (zdravá) váha."
}
```

# Podmínky

```
elseif ($bmi -le 30) {  
    Write-Host -ForegroundColor Yellow "Nadváha."  
}  
elseif ($bmi -le 35) {  
    Write-Host -ForegroundColor Orange "Obezita prvního stupně."  
}  
elseif ($bmi -le 40) {  
    Write-Host -ForegroundColor Red "Obezita druhého stupně."  
}  
else {  
    Write-Host -ForegroundColor Magenta "Morbidní obezita."  
}
```

## Ternární operátor

- Nově s verzí 7.0
- Zápis *If*, *Else* v jednom příkaze
- <podmínka> ? <if-true> : <if-false>

```
$path = "/proc/self"  
# Příkaz Test-Path zjišťuje,  
# zda v systému existuje daná cesta.  
$system = (Test-Path $path) ? "Unix" : "Windows" # Unix
```

## Switch

- Větvení programu na základě hodnoty jedné proměnné
- Lze testovat přesné hodnoty, porovnávat pomocí operátorů nebo ověřovat vůči regulárnímu výrazu
- Dá se použít i na kolekce
- Možnost Default slouží jako výchozí, když porovnání se všemi ostatními větvemi neuspělo

## Syntax

```
Switch(<proměnná>) {  
    <možnost1> {<příkazy>}  
    <možnost2> {<příkazy>}  
    # ...  
    <možnostN> {<příkazy>}  
    Default      {<příkazy>}  
}
```

- Klíčová slova *Break* a *Continue* se hodí při testování hodnot pole nebo pokud Switch obsahuje více možností odpovídajících pro danou hodnotu
  - Příkaz *Break* ukončí testování aktuální i všech následujících hodnot kolekce.
  - Příkaz *Continue* ukončí testování pouze aktuální hodnoty.
- Aktuální testovaná hodnota je uložena v proměnné `$_`

## Příklad: Switch v poli

```
Switch (1,4,-1,3,"Číslo",2,1)
{
    1 { "Lednička" }
    {$_ -lt 0} { Continue }
    {$_ -isnot [Int32]} { Break }
    {$_ % 2} {
        "$_ je liché"
    }
    {-not ($_ % 2)} {
        "$_ je sudé"
    }
}
```

## Výstup

```
Lednička
1 je liché
4 je sudé
3 je liché
```

- Podmíněné opakování příkazů
- Stejně jako u switche lze použít *Break* nebo *Continue* k předčasnému ukončení

## Cyklus For

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

- <Inicializace> umožňuje vytvořit iterační proměnné
- <Podmínka> je výraz, který musí být splněn, aby se cyklus mohl opakovat
  - Kontrola proběhne také před prvním spuštěním cyklu
- V <Opakování> jsou obsaženy všechny příkazy, které se provedou po skončení jedné iterace cyklu
- Žádná z těchto tří částí není vyžadována

## Příklad: FizzBuzz

```
for ($i = 1; $i -le 20; $i++) {  
    if ($i % 3 -eq 0 -and $i % 5 -eq 0) {  
        Write-Host "FizzBuzz," -NoNewline  
    }  
    elseif ($i % 3 -eq 0) {  
        Write-Host "Fizz," -NoNewline  
    }  
    elseif ($i % 5 -eq 0) {  
        Write-Host "Buzz," -NoNewline  
    }  
    else {  
        Write-Host "$i," -NoNewline  
    }  
}  
  
# Výstup: 1,2,Fizz,4,Buzz,Fizz,7,8,  
# Fizz,Buzz,11,Fizz,13,14,FizzBuzz,  
# 16,17,Fizz,19,Buzz,
```

## Zadání

- Výpis čísel od 1 do 20.
- Násobky 3 jsou nahrazeny slovem "Fizz"
- Násobky 5 jsou nahrazeny slovem "Buzz"
- Násobky 3 i 5 jsou nahrazeny "FizzBuzz"
- Pozn.: Příkaz `$i++` je ekvivalentem  
`$i = $i + 1`  
nebo `$i += 1`



## Příklad: Vnořené cykly Pascalův trojúhelník

```
$size = 5
$values = @(1)
for ($i = 1; $i -le $size; $i++) {
    $spaces = " " * ($size - $i)
    $row = ""
    $nextValues = @(1)
    for ($i = 0; $i -lt $values.Length; $i++) {
        # Přičtení prvku kolekce z i-té pozice
        $row += $values[$i]
        $row += " "
        if ($i + 1 -lt $values.Length) {
            $nextValues += $values[$i] + $values[$i + 1]
        }
    }
    Write-Host $spaces$row
    # Operátor += pro pole znamená
    # přidání prvku na jeho konec
    $nextValues += 1
    $values = $nextValues
}
```

## Výstup

```
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
```

## Cyklus ForEach

- Průchod přes kolekci

```
foreach ($<položka> in $<kolekce>) {  
    <příkazy>  
}
```

## Příklad

- Vypsání informací o všech souborech větších než 100 KB v aktuální složce

```
foreach ($soubor in Get-ChildItem) {  
    if ($soubor.length -gt 100KB) {  
        Write-Host $soubor  
        Write-Host $soubor.Length  
        Write-Host $soubor.LastAccessTime  
    }  
}
```

## Cyklus While

- Stejný jako cyklus For bez inicializace a příkazů po skončení
- Hodí se, když nepotřebujeme vytvářet další proměnné nebo nechceme provádět některé příkazy (třetí část For) po zavolání *Continue*

```
<Inicializace>  
while (<Podmínka>) {  
    <příkazy>  
    # ...  
    <Opakování>  
}
```

## Příklad: největší společný dělitel dvou přirozených čísel (Euklidův algoritmus)

```
[int]$a = Read-Host "Zadejte první číslo"
[int]$b = Read-Host "Zadejte druhé číslo"
```

```
while($b -ne 0){
    $rem = $a % $b
    $a = $b
    $b = $rem
}
```

```
Write-Host "Největší společný dělitel:" $a
```

## Cyklus Do

- Nejdříve vykoná příkazy, podmínky kontroluje až potom
- Dvě varianty: **Do While**, **Do Until** - podmínka je negovaná
- Vhodné při zpracování vstupu (nejdříve načíst, až pak ověřit)

<Inicializace>

```
do {  
    <příkazy>  
    # ...  
    <Opakování>  
}  
while (<Podmínka>)
```

## Příklad: Hádání čísla

```
$target = Get-Random -Maximum 100  # Náhodné číslo mezi 0 a 99
$numTries = 0

do {
    $numTries++
    [int]$guess = Read-Host "Zadejte číslo"
    if ($guess -lt $target) {
        Write-Host "Více."
    }
    elseif ($guess -gt $target) {
        Write-Host "Méně."
    }
} while ($guess -ne $target)

Write-Host "Trefa! Uhádli jste po $numTries pokusech"
```

# Pipeline

- Řetězení několika příkazů na jednom řádku (*one-linery*)
- Předávání výstupů mezi příkazy bez nutnosti vytváření proměnných na mezivýsledky
- K aktuálnímu výstupu příkaz přistupuje ve speciálním (vyhrazeném) objektu `$_`
  - V dokumentaci jednotlivých cmdletů je nutné dohledat, v jakých vlastnostech skončí daný vstup
- Zapisuje se operátorem `|` (pipe, roura)
- Jestliže je výsledný pipeline moc dlouhý, doporučuje se rozdělit ho kvůli přehlednosti na více řádků končící (nebo začínající) rourou
  - Umožňuje případné komentování jednotlivých částí

## Použití s pipeline

```
Get-Process |  
Sort-Object Name |  
Format-Table -Property Name, CPU
```

## Použití bez pipeline

```
$Processes = Get-Process  
$SortedPS = Sort-Object Name  
Format-Table $SortedPS -Property Name, CPU
```

## Where-Object

- Výběr prvků v kolekci, jejichž vlastnosti splňují danou podmínku
- Kromě ScriptBlocku lze také porovnávat přímo v parametrech cmdletu
- Často se používá alias `where`

Příklad: *Výběr procesů s nízkou prioritou* (ekvivalentní způsoby)

- 1 `Get-Process | Where-Object PriorityClass -eq -Value "BelowNormal"`
  - 2 `Get-Process | Where-Object -Property PriorityClass -eq "BelowNormal"`
  - 3 `Get-Process | where {$_.PriorityClass -eq "BelowNormal"}`
- Dá se použít také na ověření existence dané vlastnosti

Příklad: *Výběr všech adresářů v aktuální složce*

```
Get-ChildItem | where {$_.PSIsContainer}
```



# Filtrování v pipeline

## Select-Object

Funkce:

- Výběr vlastností v každém prvku: `-Property`  
`Get-Process | Select-Object -Property ProcessName, Id, WS`
- Filtrování jedinečných hodnot: `-Unique`  
`"a", "b", "c", "a", "c", "a" | Select-Object -Unique`  
*# výstup: a b c*
- Upřesnění rozmezí:
  - jednotlivé pozice (`-Index <Seznam>`)  
`$a = Get-EventLog -LogName "Windows PowerShell"`  
*# výběr prvního a posledního záznamu*  
`$a | Select-Object -Index 0, ($a.count - 1)`

# Filtrování v pipeline

- prvních (-First)  $N$  prvků

*# Prvních 5 řádků v souboru*

```
Get-Content file.txt | select -First 5
```

- posledních (-Last)  $N$  prvků

*# Poslední 3 řádky ze schránky*

```
Get-Clipboard | select -Last 3
```

- přeskočení  $N$  prvků (-Skip, -SkipLast) nebo vybraných pozic (-SkipIndex)

*# Vytvoření vzdáleného sezení na všech serverech*

*# kromě prvního serveru v souboru*

```
New-PSSession -ComputerName (Get-Content Servers.txt |  
Select-Object -Skip 1)
```

- Pozn.: Při použití -First nebo -Index se vstup přestane generovat (nebo zpracovávat) ve chvíli, kdy se načtou všechny požadované prvky

- K vypnutí této optimalizace je třeba upřesnit přepínač -Wait

# Filtrování v pipeline

## Sort-Object

- Setřídění kolekce na základě vlastností
- Výchozím parametrem řazení je atribut **Name**
- Lze upřesnit získání pouze prvních (**-Top**) nebo posledních (**-Bottom**) *N* prvků z výsledné struktury

```
Get-ChildItem "~/config" | Sort-Object -Top 5
```

```
<#
```

```
Directory: /home/auburn/.config
```

<i>Mode</i>	<i>LastWriteTime</i>	<i>Length</i>	<i>Name</i>
----	-----	-----	----
d----	5/23/2020 3:23 PM		alacrity
d----	5/9/2020 11:03 PM		asciinema
d----	3/28/2020 3:00 PM		autostart
d----	5/3/2020 2:24 PM		blender
d----	4/21/2020 11:03 PM		calcurse

```
#>
```

# Filtrování v pipeline

- Prvky lze řadit buď vzestupně (výchozí způsob), nebo sestupně (-Descending)

```
Get-History | Sort-Object -Property Id -Descending
```

```
<#
```

```
Id CommandLine
```

```
-- -----
```

```
10 Get-Command Sort-Object -Syntax
```

```
9 $PSVersionTable
```

```
8 Get-Command Sort-Object -Syntax
```

```
7 Get-Command Sort-Object -ShowCommandInfo
```

```
6 Get-ChildItem -Path ~/Pictures | Sort-Object -Property Length
```

```
5 Get-Help Clear-History -online
```

```
4 Get-Help Clear-History -full
```

```
3 Get-ChildItem | Get-Member
```

```
2 Get-Command Sort-Object -Syntax
```

```
1 Set-Location ~/Documents
```

```
#>
```

- Opět možno vybírat pouze jedinečné hodnoty přepínačem **-Unique**
- Parametr **-Stable** zaručí, že rovnající se hodnoty budou ve výsledku mít stejné pořadí jako v původní struktuře
- Vlastní styl porovnávání se dá definovat ve **ScriptBlocku** (výchozí porovnává na základě typu vlastností)

# Filtrování v pipeline

## Příklad: Řazení na základě zbytku po dělení

### Výchozí třídění

```
1..17 | Sort-Object {$_ % 3}
```

9  
15  
3  
12  
6  
13  
10  
16  
1  
7  
4  
11  
5  
14  
2  
8  
17

### Stabilní třídění

```
1..17 | Sort-Object {$_ % 3} -Stable
```

3  
6  
9  
12  
15  
1  
4  
7  
10  
13  
16  
2  
5  
8  
11  
14  
17

- Samostatný blok kódu
  - Po definici se nespustí sám, je třeba ho zavolat
- Název funkce by měl dodržovat standardní formát Cmdletu
- Lze ji zadefinovat s parametry, případně v ní vrátet hodnotu pro použití v jiných příkazech nebo předčasné skončení
- Členění příkazů na menší logické celky pomůže zvýšit přehlednost a čitelnost kódu
- Snižuje potřebu kopírování kódu

## Základní syntax

```
function <název-funkce> {<příkazy>}
```

## Příklad

```
function Get-Weekday { (Get-Date).DayOfWeek }
```

- Část funkce v { } se nazývá **ScriptBlock** (typ)
- Lze ji využít anonymně (bez názvu) v pipelinech

## Příklad: ScriptBlock u WhereObject

```
Get-ChildItem -Path *.txt | where {$_ .length -gt 10000}
```



## Parametry

- Proměnné vnášené do funkce

### *Pojmenované parametry*

```
Get-ChildItem -Path "~/.scripts" -Depth 2
```

- Jsou zadány jménem, popř. typem
- Powershell umožňuje odkazovat se na ně jejich částečným názvem (mimojiné podporuje doplňování tabulátorem), pokud je ovšem jednoznačný
  - Vhodné pouze pro rychlé testování v konzoli

## Příklad: částečné názvy

### Nejednoznačný název

```
PS /home/auburn> Get-ChildItem -P "~/.scripts"
```

Get-ChildItem: Parameter cannot be processed because the parameter name 'P' is ambiguous. Possible matches include: -Path -PipelineVariable -LiteralPath.

### Jednoznačný název

```
PS /home/auburn> Get-ChildItem -Pa "~/.scripts"
```

```
Directory: /home/auburn/.scripts
```

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-----	5/6/2020 1:29 AM	1165	btw,

- Možnost přiřadit také výchozí hodnotu (využije se, když nebude při volání funkce určena)
- Dva způsoby zápisu (ekvivalentní, ale styl vlevo se v Powershellu používá více)

## Zápis ve ScriptBlocku s param

```
function <název> {  
    param([typ]$parametr1,  
        [typ]$parametr2,  
        ...,  
        [typ]$parametrN)  
  
    <příkazy>  
}
```

## Zápis podobný u běžných prog. jazyků (C++, Java)

```
function <název> ([typ]$parametr1,  
                [typ]$parametr2,  
                ...,  
                [typ]$parametrN)  
{  
    <příkazy>  
}
```

## Příklad: Funkce s výchozí hodnotou

### Soubory menší než Size

```
function Get-SmallFiles {  
    param (  
        $Size = 100,  
        $Dir = $HOME  
    )  
    Get-ChildItem $Dir | Where-Object {  
        # všechny objekty menší než $Size, které nejsou složkami  
        $_.Length -lt $Size -and !$_.PSIsContainer  
    }  
}
```

- Lze zavolat např.

```
Get-SmallFiles -Dir "." # aktuální složka  
# ~ je zkratka pro domovský adresář uživatele na Linuxu  
Get-SmallFiles -Dir "~/Pictures" -Size 10000  
# nejmenované parametry musí být ve stejném pořadí jako v param  
Get-SmallFiles 1024 "~/.config"
```

## *Poziční parametry*

- Nepoužívají se tak často jako pojmenované parametry, protože ty lze v případě potřeby na poziční snadno převést
- Do funkce se vnáší pouze beze jména v pevně daném pořadí (ale mohou být prokládány jmenovanými parametry, čímž se volání může stát nepřehledným)
- Uložené v proměnné `$args` (pole)

## **Příklad: Zápis do souboru ze schránky**

```
function Write-Clipboard {  
    Get-Clipboard | Set-Content -Path $args[0]  
}
```

*# Použití:*

```
Write-Clipboard vystup.out
```

## Povinné parametry

- Upřesňují se v atributu `Parameter`
  - Zde se dá nastavit také pozice samotného parametru, čímž se jmenovaný parametr zkombinuje s pozičním
- Jestliže se funkce zavolá bez nich, bude se vyžadovat jejich doplnění (interaktivně od uživatele)

## Příklad: Vyhledávání souboru

```
function Find-File {  
    param(  
        $Dir, # nepovinný jmenovaný parametr  
        [Parameter(Mandatory, Position=0)]  
        [string]$Name # povinný poziční parametr  
    )  
    Get-ChildItem $Dir -Recurse -Filter $Name -File  
}
```

## Přepínače

- Jmenované parametry s hodnotami typu *Boolean* (\$true,\$false)
- K přiřazení hodnoty \$true stačí při volání zapsat pouze jejich název
- Definují se typem [switch]

## Příklad

```
function Write-Clipboard {  
    param(  
        [string]$Path,  
        [switch]$Debug  
    )  
    if ($Debug) {  
        Get-Clipboard  
    }  
    Get-Clipboard |  
    Set-Content -Path $Path  
}
```

### Volání pro Debug=\$true

```
Write-Clipboard -Debug -Path a.out  
Write-Clipboard -Debug:$true -Path a.out
```

### Volání pro Debug=\$false

```
Write-Clipboard -Path a.out  
Write-Clipboard -Debug:$false -Path a.out
```

# Pipeline ve funkcích

- Pokročilejší ovládání vstupu z pipeline
- Využití při průchodu přes kolekci

## Syntax

```
function <název> {  
    begin {<příkazy>}  
    process {<příkazy>}  
    end {<příkazy>}  
}
```

- Části *Begin*, *End* se spouští pouze jednou - při první či poslední zpracované položce
- Část *Process* proběhne pro každou položku kolekce



# Pipeline ve funkcích

- Během volání funkce je dostupná speciální proměnná `$input`, ve které je uložen dosud zpracovaný vstup
  - V *Begin* tedy bude prázdná, a v *End* bude obsahovat původní objekt poslaný přes pipeline
  - Jestliže je definován blok *Process*, hodnoty se přesouvají do proměnné `$_`, a `$input` tak bude vždy prázdný
  - Lze ji použít i mimo bloky, kde její hodnota bude stejná jako po přechzení v *End*
- Pokud je definován jakýkoliv z těchto bloků, veškerý kód mimo ně se už nebere jako kód a při zavolání funkce spadne (s dost kryptickými chybami)

```
function Get-PipelineInput {  
    process {"Aktuální objekt: $_ "  
    end {"Konec: Celý vstup: $input"}  
}
```

```
PS /home/auburn> 1,2,4 | Get-PipelineInput  
Aktuální objekt: 1  
Aktuální objekt: 2  
Aktuální objekt: 4  
Konec: Celý vstup:
```

## Filtr

- Ekvivalent funkce, jejíž kód je celý zabalený do bloku *Process*
- Definuje se klíčovým slovem *filter* na místě *function*

## Příklad

- Filtr, která zobrazuje buď celý záznam, nebo pouze zprávu (člena *Message*) z výpisu

```
filter Get-ErrorLog ([switch]$message)
{
    if ($message) { Out-Host -InputObject $_.Message }
    else { $_ }
}
```

Složitější datové struktury umožňující definici vlastních typů.

## Pojmy

- *Třída* (Class) - definice objektu
- *Instance* - konkrétní proměnná s typem tohoto objektu
- *Atribut* (Property) - proměnná uvnitř objektu
- *Metoda* - funkce definovaná pro objekt, má přístup k jeho atributům
  - Možnost definovat více metod se stejným názvem lišící se v seznamu jejich parametrů (*Method Overloading* - přetěžování metod)
- *\$this* - reference na objekt uvnitř jeho metod
- *Konstruktor* - metoda volaná při vzniku objektu, slouží k inicializaci jeho atributů
- *Statický člen* - Atribut nebo metoda existující v rámci třídy jako takové, nepatří žádné konkrétní instanci

## Příklad: vytvoření třídy

```
1 class Tea {
2     # statický atribut:
3     # celkový počet nálevů
4     static [int]$TotalInfusions = 0
5     # atributy každé instance
6     [string]$Name
7     [string]$Origin
8     [int]$Infusions
9     # konstruktory objektu
10    Tea($Name, $Origin, $Infusions){
11        $this.Name      = $Name
12        $this.Origin     = $Origin
13        $this.Infusions = $Infusions
14    }
15    Tea(){ $this.Name = 'Undefined' }
16    # metoda v instanci
17    [void] Pour(){
18        if($this.Infusions -gt 0){
19            $this.Infusions--
20            # přístup ke statickému atributu
21            [Tea]::TotalInfusions++
22        }
23    }
24 }
```

## Viditelnost atributů

- Veškeré atributy třídy jsou **veřejné**
  - Jestliže máme odkaz na instanci, můžeme v ní číst, měnit a používat úplně všechno
  - Pokud víme o existenci objektu, dostaneme se i ke všem jeho statickým atributům
  - Členové s vlastností **hidden** (nastavuje se podobně jako static) se pouze nezobrazují při doplňování tabulátorem nebo při zavolání Get-Member bez přepínače -Force

## Příklad: čajový dýchánek

*# 1. způsob nastavení atributů:*

*# Upřesnění v konstruktoru při instancování*

```
$cj = [Tea]::new('China Jasmin', 'China', 3)
```

```
$cj.Pour()
```

```
$cj.Pour()
```

*# 2. způsob: Přepsání po vytvoření*

```
$Mate = [Tea]::new()
```

```
$Mate.Name = 'Mate Atacama'
```

```
$Mate.Origin = 'Brazil'
```

```
$Mate.Infusions = 1
```

```
$Mate.Pour()
```

*# 3. způsob: Cmdlet New-Object*

```
$Rooibos = New-Object Tea -ArgumentList "Rooibos Pretoria",  
"South Africa", 1
```

```
$Rooibos.Pour()
```

```
$Rooibos.Pour()
```

```
[Tea]::TotalInfusions # 4
```

## Dědičnost

- Rozšíření vlastností objektu v další třídě
- Třída, která dědí, má automaticky přístup ke všem členům třídy původní
- Atributy rodičovské třídy se inicializují v `base()`
- Zděděné metody se mohou dle potřeby přepisovat

## Příklad

### Základová třída

```
class Beverage {  
    [double]$Volume  
    [double]$Price  
    [void] Pour() {  
        $this.Volume /= 2  
    }  
  
    Beverage([double]$Volume, [double]$Price){  
        $this.Volume = $Volume  
        $this.Price = $Price  
    }  
}
```

## Příklad

### Dědicí třída

```
class Juice : Beverage {  
    [double]$FruitPart  
    Juice($Volume, $Price, $FruitPart) : base($Volume, $Price) {  
        $this.FruitPart = $FruitPart  
    }  
    # Přetížená metoda  
    [void] Pour() {  
        $this.Volume /= 10  
    }  
    [string] ToString(){  
        return ("Juice {0}% - ${1} - {2} ml" -f $this.FruitPart,  
            $this.Price, $this.Volume)  
    }  
}  
  
$caprisonne = [Juice]::new(250, 0.5, 10)  
$caprisonne.Pour()  
$caprisonne.ToString()  
# Juice 10% - $0.5 - 25 ml
```



1 Úvod

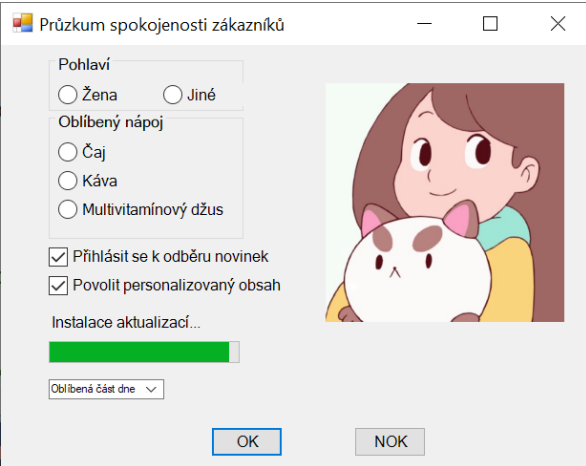
2 Syntax

3 Pokročilé

- Formuláře
- Nástroje
- Remoting

- Tvorba grafických prostředí možná pouze pro Windows
- Založeno na platformě .NET Framework a její knihovně tříd Forms
- Nelze s ním pracovat v Powershell Core
- Ovládací prvky:
  - Tlačítka: Button, CheckBox, RadioButton
  - Textový vstup: TextBox
  - Popis: Label
  - Obrázek: PictureBox
  - Ukazatel průběhu: ProgressBar
  - Rozevírací nabídka: ComboBox
  - Seskupení několika prvků: GroupBox

## Ukázka použití ovládacích prvků. Zdrojový kód dostupný [zde](#).



Průzkum spokojenosti zákazníků

Pohlaví

☐ Žena ☐ Jiné

Oblíbený nápoj

☐ Čaj

☐ Káva

☐ Multivitaminový džus


☒ Přihlásit se k odběru novinek

☒ Povolit personalizovaný obsah

Instalace aktualizací...

Oblíbená část dne

OK NOK



## Atributy ovládacích prvků

- K textovému obsahu komponent se přistupuje v atributu **Text**
  - U samotného formuláře tato vlastnost určuje název okna
  - Mimojiné jde měnit i **Font** textu, například  
`$label.Font = 'Segoe UI font,12'`
- Velikost prvku je uložena v atributu **Size** (a také **Width,Height**)
  - Dá se nastavit dvěma způsoby:
    - 1 Pomocí objektů v `System.Drawing` (třeba přidat přes `Add-Type`)  
`$form.Size = New-Object System.Drawing.Size($width,$height)`
    - 2 Deklarací v řetězci  
`$form.Size = '{0},{1}' -f $width,$height`
  - Také se může dopočítat automaticky (v závislosti na rozměru obsahu) nastavením booleovské vlastnosti **AutoSize**
- Pozice komponenty se definuje v členu **Location**
  - Udává se v pixelech
  - Vzdálenost se měří od levého horního rohu okna
  - K přiřazení se může využít `System.Drawing.Point` nebo opět řetězec

- *Atributy specifické pro formulář*

- Vlastnost **StartPosition** umožňuje zarovnání formuláře na obrazovce, nejčastěji na střed:

```
$form.StartPosition = 'CenterScreen'
```

- Boolean **TopMost** udává, zda se formulář bude zobrazovat nad všemi ostatními (i při přepnutí na jiné okno)
- Do kolekce **Controls** je třeba přidat veškeré ovládací prvky, které chceme ve formuláři zobrazit

```
1 $form.Controls.AddRange(@($label, $userInput, $genButton))
```

```
2 $form.controls.Add($label)
```

```
$form.Controls.Add($userInput)
```

```
$form.Controls.Add($genButton)
```

- Ohraničení okna se nastaví v atributu **FormBorderStyle**
  - Hodnota **'FixedDialog'** zabrání změně velikosti okna
- Po veškeré konfiguraci a přidání do formuláře dojde k jeho zobrazení metodou **ShowDialog**

## Zachytávání událostí

- K informacím o události lze v kontrolních funkcích opět přistoupit v automatické proměnné `$_`
- Ovládací funkce musí být typu `ScriptBlock`, proto pokud jsme původně deklarovali samostatnou funkci, je třeba ji při přidání obalit do `{ }`
- *Kliknutí myši*
  - Ovládací funkce se přidává pomocí metody **Add\_Click**  
`$genButton.Add_Click({Shuffle})`
- *Stisknutí klávesy*
  - Je třeba zapnout booleovský atribut formuláře **KeyPreview**
    - Ten umožní formuláři obdržet jednotlivé klávesy
    - Jinak je dostává pouze aktivní komponent (např. textové pole, do kterého se píše)

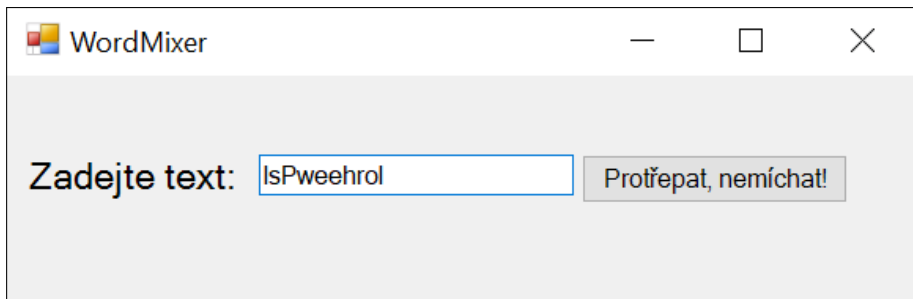
- Přidává se metodou **Add\_KeyDown**:

```
$form.Add_KeyDown({  
    if($_.KeyCode -eq 'Enter'){  
        Shuffle  
    }  
})
```

- Pro přístup k funkcím Forms je třeba načtení pomocí Add-Type  
**Add-Type -AssemblyName System.Windows.Forms**

# Ukázka formulářů

**Aplikace, která náhodně zpřehází písmenka ve vstupním řetězci.**



The screenshot shows a Windows application window titled "WordMixer". The window has a standard title bar with minimize, maximize, and close buttons. The main content area has a light gray background. On the left, the text "Zadejte text:" is displayed in a bold, black font. To its right is a text input field with a blue border, containing the text "IsPweehrol". Further to the right is a button with a gray background and black text that reads "Protřepat, nemíchat!".



# Ukázka formulářů

```
Add-Type -AssemblyName System.Windows.Forms
```

```
Add-Type -AssemblyName System.Drawing
```

```
$fontFam = 'Segoe UI font'
```

```
$fontSize = 12
```

```
# Vlastnosti formuláře
```

```
$form = New-Object System.Windows.Forms.Form
```

```
$form.Text = 'WordMixer'
```

```
$form.Size = New-Object System.Drawing.Size(600,200)
```

```
$form.StartPosition = 'CenterScreen'
```

```
$form.TopMost = $true
```

```
# Popisek textového pole
```

```
$label = New-Object System.Windows.Forms.Label
```

```
$label.Font = $fontFam + ', ' + $fontSize
```

```
$label.Location = New-Object System.Drawing.Point(10,  
                                                    ($form.Size.Height / 4))
```

```
$label.Width = 150
```

# Ukázka formulářů

```
$label.Height = $fontSize * 3
$label.Text = 'Zadejte text: '

# Textové pole pro uživatelský vstup
$userInput = New-Object System.Windows.Forms.TextBox
$userLocX = $label.Location.X + $label.Width
$userInput.Location = New-Object System.Drawing.Point($userLocX,
    $label.Location.Y)
$userInput.Font = $fontFam + ', ' + $fontSize
$userInput.Width = 200

function Shuffle {
    $userInput.Text = ($userInput.Text -split '' |
        Sort-Object {Get-Random}) -join ''
}

# Tlačítko pro generování
$genButton = New-Object System.Windows.Forms.Button
$genButton.Text = "Protřepat, nemíchat!"
```

# Ukázka formulářů

```
$genButton.AutoSize = $true
$buttonLocX = $userInput.Location.X + $userInput.Width + 5
$genButton.Location = New-Object System.Drawing.Point($buttonLocX,
    $label.Location.Y)
$genButton.Add_Click({Shuffle})

$form.KeyPreview = $true
# Možnost stisknout také Enter pro novou generaci
$form.Add_KeyDown({
    if($_.KeyCode -eq 'Enter'){
        Shuffle
    }
})

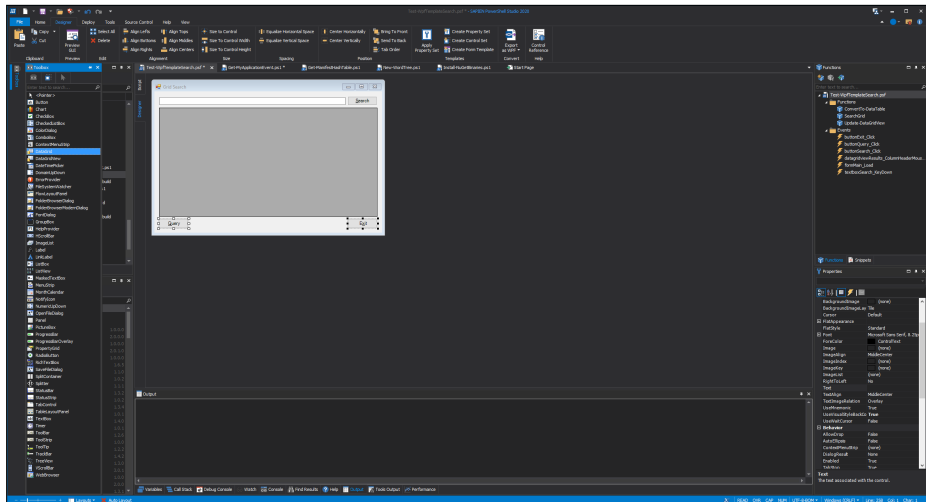
$form.controls.Add($label)
$form.Controls.Add($userInput)
$form.Controls.Add($genButton)
$form.ShowDialog()
```

## Powershell Studio

- Vzhledově podobné Visual Studiu
- Obsahuje grafický návrhář a debugger
- Převod skriptů na spustitelné (.exe) soubory
- Vytváření instalačních (.msi) skriptů
- Tvorba modulů
- Integrovaná konzole
- Monitorování výkonu a využití paměti
- Podpora klasického Powershellu a Powershell Core
- Placené (**\$399**)

# Nástroje pro tvorbu GUI

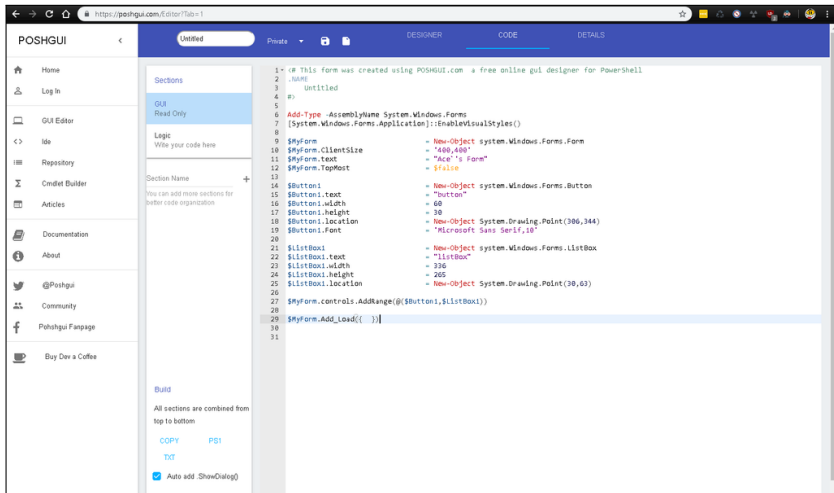
## Powershell Studio



## POSHGUI

- Online nástroj pro výrobu grafických prostředí v Powershellu
- Obsahuje repozitář uživatelských projektů
- Možnost grafické tvorby Cmdletů
- Zdarma

## POSHGUI



Spouštění kódu na vzdálených zařízeních (dostupných po síti).

## Windows Powershell

- Pouze pro Windows
- Spojení přes WinRM (Windows Remote Management)

## Powershell Core

- Multiplatformní
- Komunikace pomocí SSH (Secure Shell)
- Snazší konfigurace zabezpečení (netřeba nastavovat SSL/TLS certifikáty kvůli HTTPS)



# Konfigurace SSH na Windows

OpenSSH - Open-source implementace SSH

## Instalace OpenSSH z Powershellu (jako Administrátor)

- 1 Zjištění dostupnosti balíčků

```
Get-WindowsCapability -Online | ? Name -like 'OpenSSH*'
```

Mělo by vypsát (pokud již není nainstalované):

```
Name : OpenSSH.Client~~~~0.0.1.0
```

```
State : NotPresent
```

```
Name : OpenSSH.Server~~~~0.0.1.0
```

```
State : NotPresent
```

## 2 Instalace klienta a serveru

```
Add-WindowsCapability -Online -Name OpenSSH.Client~~~~0.0.1.0
```

```
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
```

Oba příkazy by měly vrátit:

```
Path :
```

```
Online : True
```

```
RestartNeeded : False
```

# Konfigurace SSH na Windows

## 3 Automatické spouštění ssh

```
Start-Service sshd
```

```
Set-Service -Name sshd -StartupType 'Automatic'
```

## 4 Konfigurace firewallu pro server

```
New-NetFirewallRule -Name sshd -DisplayName 'OpenSSH
```

```
→ Server (sshd)' -Enabled True -Direction Inbound
```

```
→ -Protocol TCP -Action Allow -LocalPort 22
```

## 5 Úprava konfiguračního souboru

\$env:ProgramData\ssh\sshd\_config (např. přes notepad.exe  
pořád v Powershellu) - přidání dvou řádků

# Konfigurace SSH na Windows

```
# 1) Povolí ověřování heslem  
PasswordAuthentication Yes
```

```
# 2) Vytvoří subsystém (sadu příkazů, přes kterou se klient bude  
# pohodlně připojovat na server) pro Powershell.  
# Počítá se s výchozí cestou instalace pwsh  
# (jinak je třeba dosadit vlastní)  
Subsystem powershell c:/progra~1/powershell/7/pwsh.exe -sshs -NoLogo -NoProfile
```

## 6 Restartování služby sshd

```
Restart-Service sshd
```

# Konfigurace SSH na Linuxu

- 1 OpenSSH je dostupné ve správci balíčků jako `openssh` nebo `openssh-server` + `openssh-client` (liší se u různých distribucí)
  - Příklad pro Ubuntu:

```
sudo apt install openssh-client
sudo apt install openssh-server
```
- 2 Úprava konfiguračního souboru `/etc/ssh/sshd_config` - přidání dvou řádků

`PasswordAuthentication Yes`

# Umístění powershellu lze zjistit příkazem "whereis pwsh"  
`Subsystem powershell /usr/bin/pwsh -sshs -NoLogo -NoProfile`

- 3 Restartování služby `sshd`
  - Např. přes `sudo systemctl restart sshd`
  - Permanentní zapnutí přes: `sudo systemctl enable sshd`

# Remoting přes Powershell

## Vytvoření nového sezení (uložení do proměnné)

```
$session = New-PSSession -HostName <host-ip> -UserName <host-username>
```

```
PS /home/auburn> $session = New-PSSession -HostName 192.168.0.103  
↪ -UserName krist  
krist@192.168.0.103's password:  
PS /home/auburn> $session
```

Id	Name	Transport	ComputerName	ComputerType	State
--	----	-----	-----	-----	-----
2	Runspace1	SSH	192.168.0.103	RemoteMachine	Opened

## Ukončení sezení

```
Remove-PSSession -Session $session
```

## Použití sezení v příkazech

- Jde to i bez proměnné, ale u každého příkazu by byla potřeba znovu určovat adresu serveru

# Remoting přes Powershell

## Vzdálené spouštění příkazů

```
Invoke-Command $session -ScriptBlock { <kód> }
```

```
PS /home/auburn> Invoke-Command $Session -ScriptBlock { Get-Process powershell }
```

NPM(K)	PM(M)	WS(M)	CPU(s)	Id	SI	ProcessName	PSComputerName
-----	-----	-----	-----	--	--	-----	-----
28	56.29	58.20	4.38	9964	1	powershell	192.168.0.103

## Lokální proměnné

- Ve ScriptBlocku se označí prefixem \$Using:
- Zde je nelze nijak měnit, pouze referencovat

```
PS /home/auburn> $ps = "*PowerShell*"
```

```
PS /home/auburn> Invoke-Command -Session $session -ScriptBlock {  
    Get-WinEvent -LogName $Using:ps  
}
```

# Remoting přes Powershell

## *Interaktivní sezení*

```
Enter-PSSession -Session $session
```

nebo

```
Enter-PSSession -HostName <host-ip> -UserName <host-username>
```

```
PS /home/auburn> Enter-PSSession -Session $session  
[krist@192.168.0.103]: PS C:\Users\krist>
```



# Remoting přes Powershell

```
Administrator: Windows PowerShell x Administrator: PowerShell x + v
PS C:\Users\krist> Enter-PSSession -HostName 192.168.0.106 -UserName auburn
auburn@192.168.0.106's password:
[auburn@192.168.0.106]: PS /home/auburn> neofetch

,=::!!t3Z3z.,      Model: G3 3579
:tt:::tt333EE3      Uptime: 2 hours, 18 mins
Et:::ztt33EEEL @Ee.,  Shell: zsh 5.8
;tt:::tt333EE7 ;EEEEEttttt33# Resolution: 1920x1080
:Et:::zt333EEQ. $EEEEEttttt33QL CPU: Intel i5-8300H (8) @ 4.000GHZ
it:::tt333EEF @EEEEEttttt33F GPU: NVIDIA GeForce GTX 1050 Ti Mobile
;3=*^""*4EEV :EEEEEttttt33@ GPU: Intel UHD Graphics 630
,=:::!!t=.,  @EEEEEtttt33QF Memory: 3406MiB / 7822MiB
;:::ztt33) "4EEEEtttji3P* GPU Driver: Dell UHD Graphics 630 (Mobile) [1028:086f]
:t:::ztt33.:Z3z.. `` ,..g. CPU Usage: 11%
i:::ztt33F AEEEEtttt:::ztF Battery0: 100% [Full]
;:::ztt33V ;EEEEtttt:::t3 Local IP: 192.168.0.106
E:::ztt33L @EEEEtttt:::z3F
{3=*^""*4E3) ;EEEEtttt:::tz`
      ' :EEEEtttt:::z7
      "VEzjt;;z>*"

[auburn@192.168.0.106]: PS /home/auburn> |
```

Děkuji za pozornost!

- [1] Powershell Community. *About Arithmetic Operators*. URL: [https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\\_arithmetic\\_operators](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_arithmetic_operators) (cit. 15.05.2020).
- [2] Powershell Community. *About Classes*. URL: [https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\\_classes](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_classes) (cit. 15.05.2020).
- [3] Powershell Community. *About Functions*. URL: [https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\\_functions](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions) (cit. 15.05.2020).
- [4] Powershell Community. *About If*. URL: [https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\\_if](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_if) (cit. 15.05.2020).
- [5] Powershell Community. *About Switch*. URL: [https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about\\_switch](https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_switch) (cit. 15.05.2020).

- [6] Powershell Community. *Approved Verbs for PowerShell Commands*. URL: <https://docs.microsoft.com/cs-cz/powershell/scripting/developer/cmdlet/approved-verbs-for-windows-powershell-commands> (cit. 15.05.2020).
- [7] Powershell Community. *Installation of OpenSSH For Windows Server 2019 and Windows 10*. URL: [https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh\\_install\\_firstuse](https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse) (cit. 15.05.2020).
- [8] Powershell Community. *PowerShell remoting over SSH*. URL: <https://docs.microsoft.com/cs-cz/powershell/scripting/learn/remoting/ssh-remoting-in-powershell-core> (cit. 15.05.2020).
- [9] Powershell Community. *Select-Object*. URL: <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-object> (cit. 28.05.2020).

# Odkazy

- [10] Powershell Community. *Sort-Object*. URL: <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/sort-object> (cit. 28. 05. 2020).
- [11] Powershell Community. *Where-Object*. URL: <https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/where-object> (cit. 28. 05. 2020).
- [12] Rudy Mens. *Powershell GUI - How to get started*. URL: <https://lazyadmin.nl/powershell/powershell-gui-howto-get-started/> (cit. 15. 05. 2020).
- [13] Microsoft. *Introducing Windows Terminal*. URL: <https://devblogs.microsoft.com/commandline/introducing-windows-terminal/> (cit. 15. 05. 2020).
- [14] Brien Posey. *PowerShell Core vs. PowerShell: What are the differences?* URL: <http://techgenix.com/powershell-core/> (cit. 15. 05. 2020).
- [15] Wolfgang Sommergut. *PowerShell loops: For, Foreach, While, Do-Until, Continue, Break*. URL: <https://4sysops.com/archives/powershell-loops-for-foreach-while-do-until-continue-break/> (cit. 15. 05. 2020).

# Odkazy

- [16] SS64. *Powershell Data Types*. URL:  
<https://ss64.com/ps/syntax-datatypes.html> (cit. 15.05.2020).
- [17] Sapien Technologies. *PowerShell Studio 2020*. URL:  
[https://www.sapien.com/software/powershell\\_studio](https://www.sapien.com/software/powershell_studio) (cit. 15.05.2020).