



Доброго времени суток дорогие читатели. В данной статье мы познакомимся с основами **Powershell**. Данный язык программирования используется во всех ОС Microsoft начиная с Windows XP SP3. Писать Powershell скрипты должен уметь каждый уважающий себя системный администратор windows.

Все команды в Powershell как правило используются в форме командлетов. Все командлеты это специализированные классы **.NET Framework** и **.NET Core** (используется в **PowerShell Core 6** и выше).

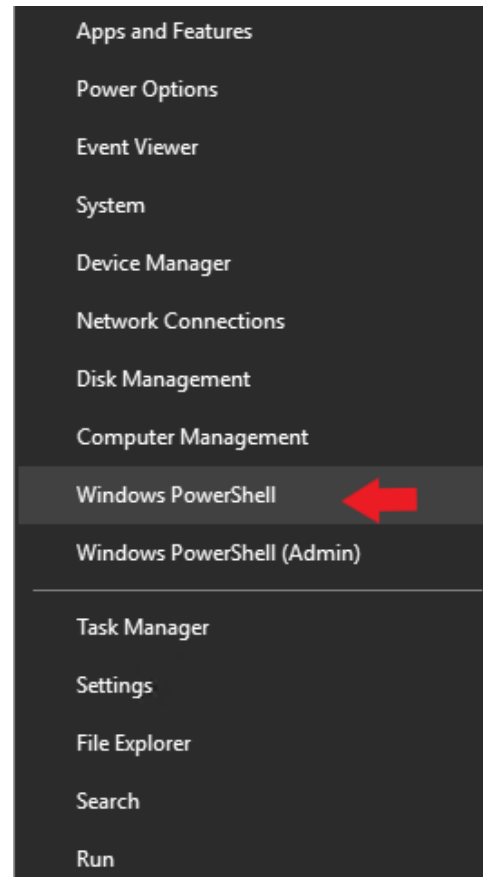
Powershell класса .NET Framework это версии с 1 по 5.1 а Powershell .NET Core это версия 6 и выше (на данный момент 7.0). По заявлению Microsoft новых функций в Powershell 5.1 (.NET Framework) вносить уже не будут. Однако 7 версия еще не полностью поддерживает все модули предыдущих версий. Но судя по всему Microsoft стремится к этому и скоро версия Core будет единственной. В общем **cmd** отходит в прошлое и теперь без Powershell никуда. Давайте приступим к практике. Так всегда лучше запоминается материал.

Запуск Powershell

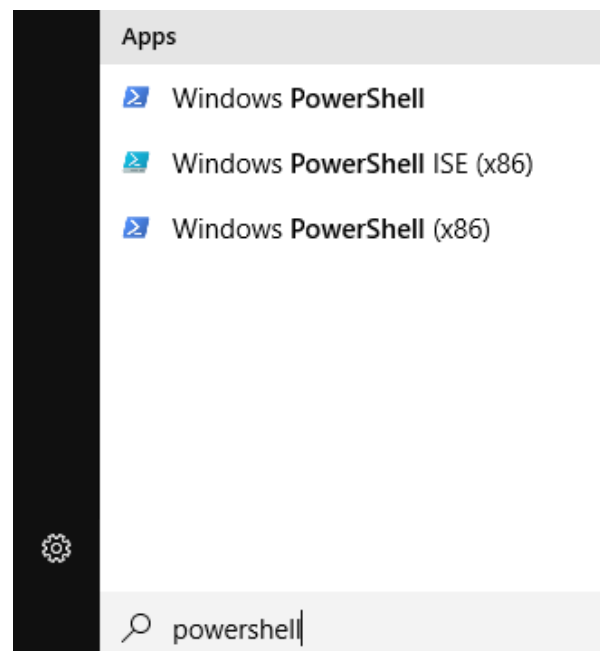
На примере **Windows 10** Powershell можно запустить просто нажав правой кнопкой мыши на меню пуск.

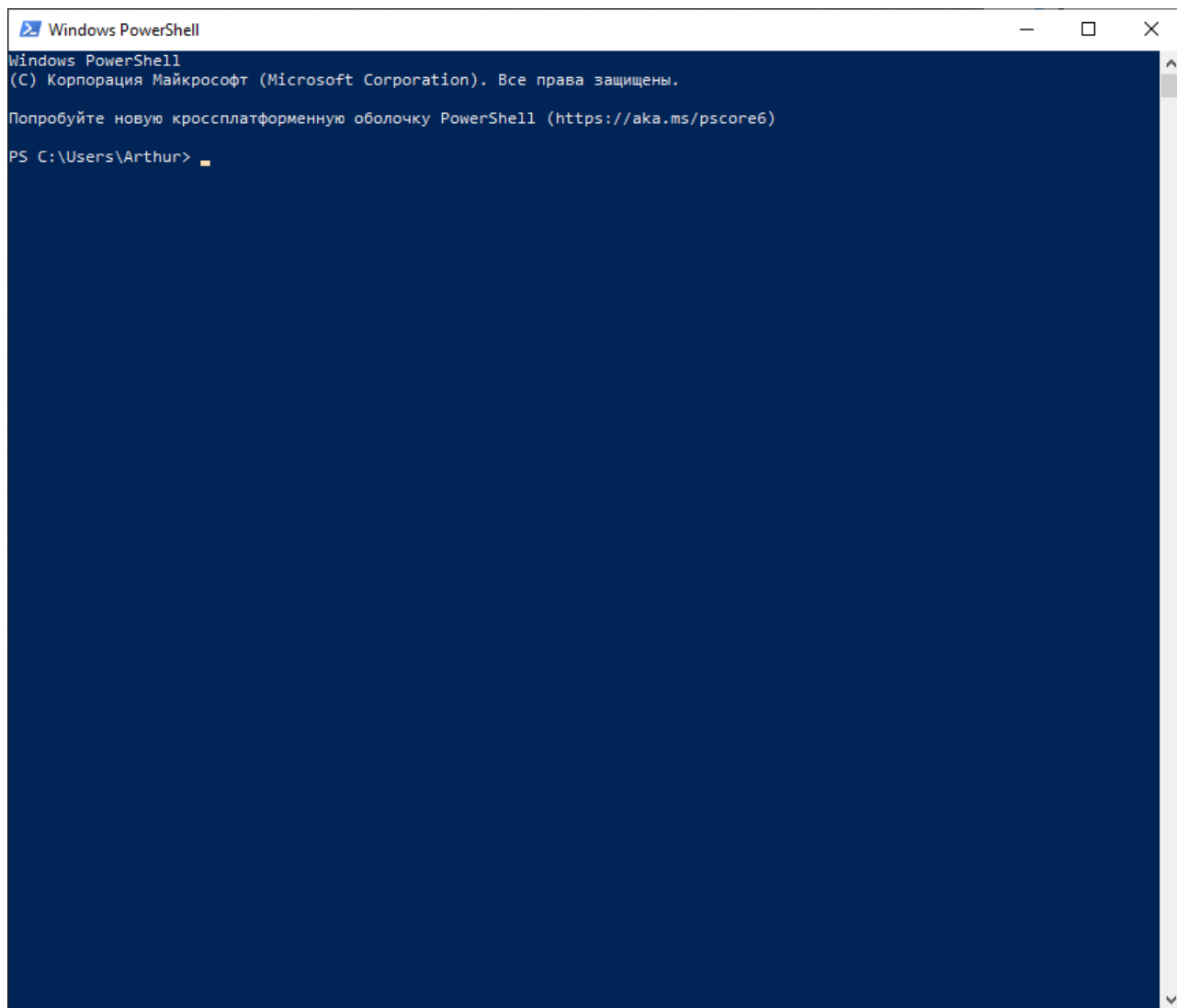
Также нижняя строчка позволяет запустить Powershell с повышенными правами администратора.

Еще можно воспользоваться поиском в Windows 10 и ввести название powershell



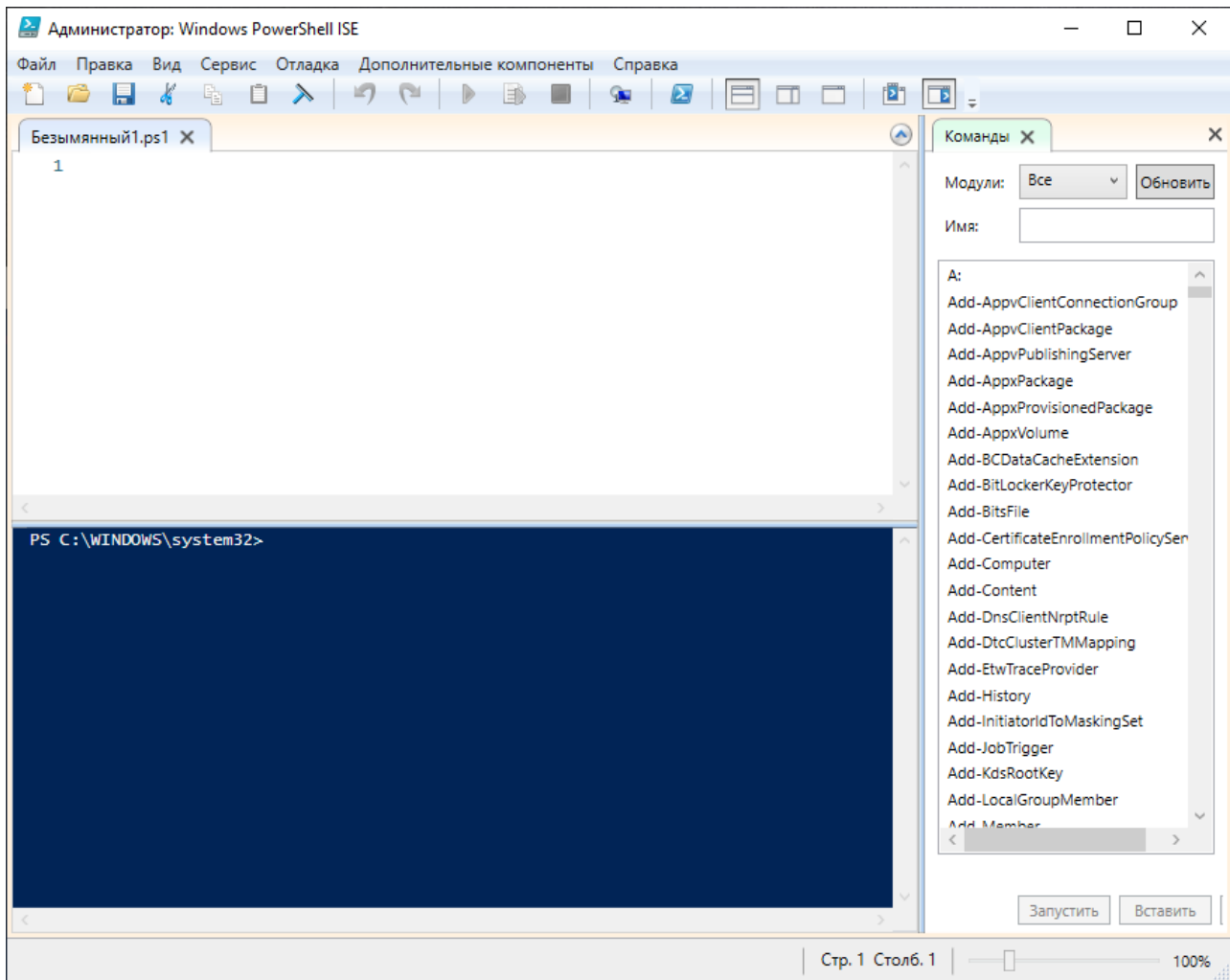
Как видно на картинке выше нашелся не только Powershell но и **Powershell ISE**. Консоль powershell удобна если требуется запустить последовательно не больше одной команды. Либо несколько команд в конвейере. Однако в случае написания полноценных скриптов лучше использовать **Powershell ISE**. Это бесплатная среда разработки сценариев на языке Powershell поставляется вместе с ОС Windows.





```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.
Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/powershell)
PS C:\Users\Arthur>
```

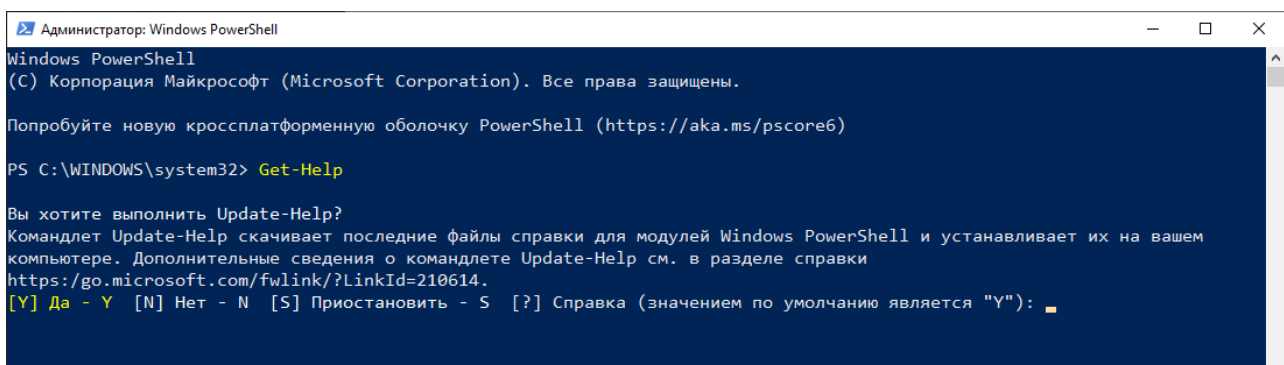
Окно Powershell



Окно Powershell ISE

Сразу после запуска консоли рекомендую запустить командлет Get-Help – встроенная справка по всем командлетам, аналог *man* в Linux.

## 1 Get-Help



Get-Help

Видим что консоль предлагает обновить встроенную помощь. Нажимаем **Y** и соглашаемся.

Командлеты

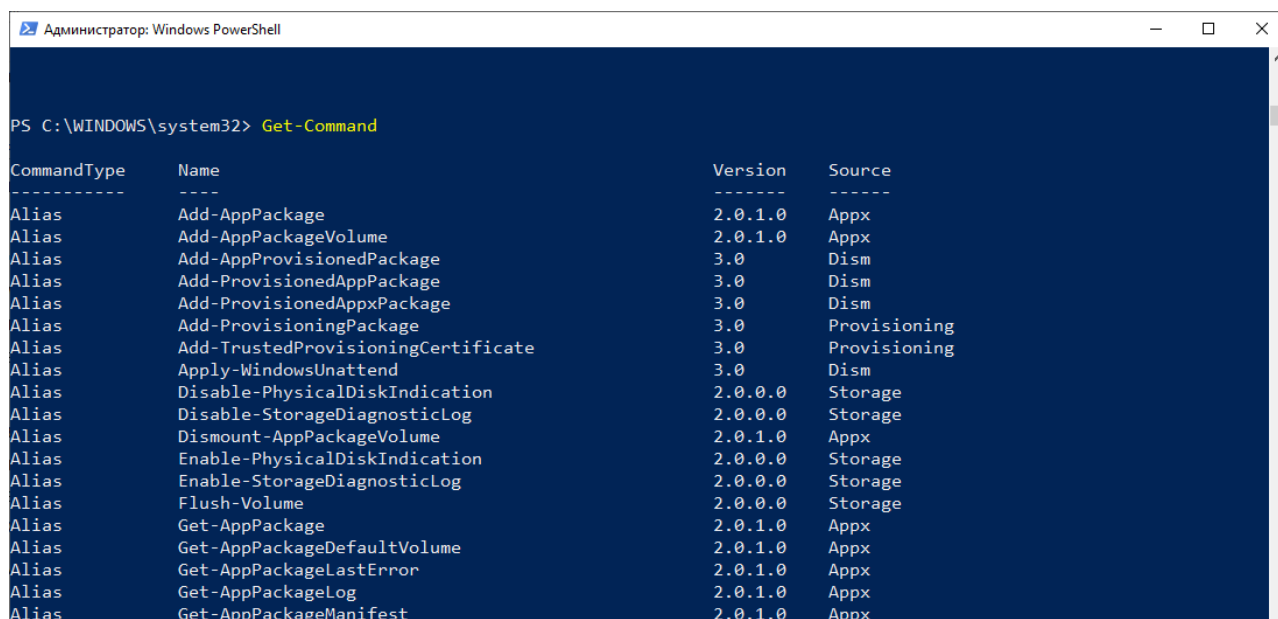
**Командлеты** – это название команд в Powershell. Каждый командлет реализует заложенную в него функциональность. Как правило наименование командлета состоит из пары: *глагол-существительное*. Например: **Get-Help** – получить помощь. Обычно **Get** используется чтобы

получить информация, **Set** – внести изменение, **New** – создать новый объект, политику и т.п. и **Remove** -удалить объект, политику и т.п.

Командлеты не чувствительны к регистру. Написать **Get** или **get** не важно, powershell воспримет эти команды одинаково.

Чтобы получить список всех доступных командлетов необходимо использовать **Get-Command**

#### 1 Get-Command



```
Администратор: Windows PowerShell

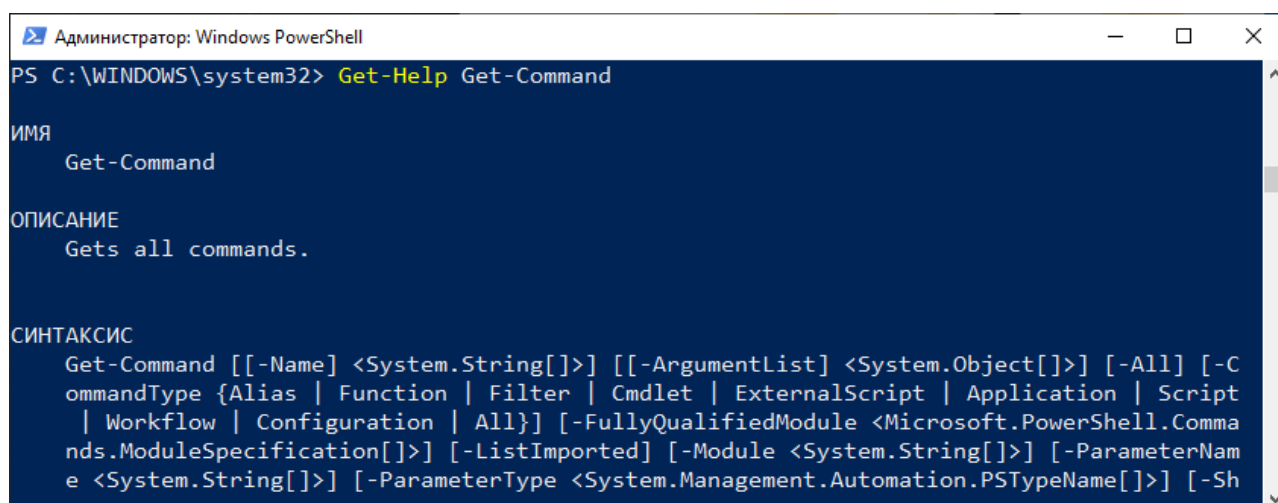
PS C:\WINDOWS\system32> Get-Command

CommandType      Name                                     Version      Source
-----
Alias            Add-AppPackage                         2.0.1.0      Appx
Alias            Add-AppPackageVolume                  2.0.1.0      Appx
Alias            Add-AppProvisionedPackage             3.0          Dism
Alias            Add-ProvisionedAppPackage              3.0          Dism
Alias            Add-ProvisionedAppxPackage             3.0          Dism
Alias            Add-ProvisioningPackage                3.0          Provisioning
Alias            Add-TrustedProvisioningCertificate     3.0          Provisioning
Alias            Apply-WindowsUnattend                  3.0          Dism
Alias            Disable-PhysicalDiskIndication          2.0.0.0      Storage
Alias            Disable-StorageDiagnosticLog            2.0.0.0      Storage
Alias            Dismount-AppPackageVolume              2.0.1.0      Appx
Alias            Enable-PhysicalDiskIndication           2.0.0.0      Storage
Alias            Enable-StorageDiagnosticLog             2.0.0.0      Storage
Alias            Flush-Volume                           2.0.0.0      Storage
Alias            Get-AppPackage                         2.0.1.0      Appx
Alias            Get-AppPackageDefaultVolume            2.0.1.0      Appx
Alias            Get-AppPackageLastError                2.0.1.0      Appx
Alias            Get-AppPackageLog                      2.0.1.0      Appx
Alias            Get-AppPackageManifest                 2.0.1.0      Appx
```

Get-Command

Для получения справки по любому командлету напишите **Get-Help имя-командлета**.  
Например

#### 1 Get-Help Get-Command



```
Администратор: Windows PowerShell

PS C:\WINDOWS\system32> Get-Help Get-Command

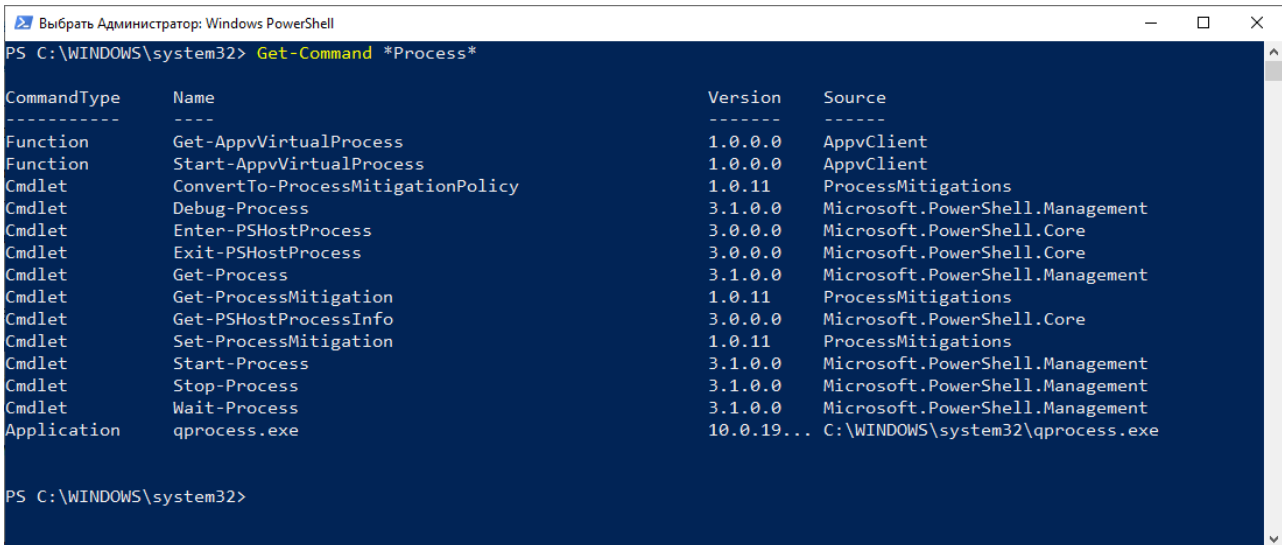
ИМЯ
    Get-Command

ОПИСАНИЕ
    Gets all commands.

СИНТАКСИС
    Get-Command [[-Name] <System.String[]>] [[-ArgumentList] <System.Object[]>] [-All] [-C
    ommandType {Alias | Function | Filter | Cmdlet | ExternalScript | Application | Script
    | Workflow | Configuration | All}] [-FullyQualifiedModule <Microsoft.PowerShell.Comma
    nds.ModuleSpecification[]>] [-ListImported] [-Module <System.String[]>] [-ParameterNam
    e <System.String[]>] [-ParameterType <System.Management.Automation.PSTypeName[]>] [-Sh
```

Давайте представим что нам необходимо вывести список командлетов для управления процессами. Воспользуемся Get-Command и укажем ему параметры для более точного поиска.

## 1 Get-Command \*Process\*



```
Выбрать Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Get-Command *Process*

CommandType      Name                                Version      Source
-----
Function         Get-AppvVirtualProcess             1.0.0.0      AppvClient
Function         Start-AppvVirtualProcess           1.0.0.0      AppvClient
Cmdlet           ConvertTo-ProcessMitigationPolicy  1.0.11       ProcessMitigations
Cmdlet           Debug-Process                     3.1.0.0      Microsoft.PowerShell.Management
Cmdlet           Enter-PSHostProcess               3.0.0.0      Microsoft.PowerShell.Core
Cmdlet           Exit-PSHostProcess               3.0.0.0      Microsoft.PowerShell.Core
Cmdlet           Get-Process                      3.1.0.0      Microsoft.PowerShell.Management
Cmdlet           Get-ProcessMitigation             1.0.11       ProcessMitigations
Cmdlet           Get-PSHostProcessInfo            3.0.0.0      Microsoft.PowerShell.Core
Cmdlet           Set-ProcessMitigation            1.0.11       ProcessMitigations
Cmdlet           Start-Process                    3.1.0.0      Microsoft.PowerShell.Management
Cmdlet           Stop-Process                     3.1.0.0      Microsoft.PowerShell.Management
Cmdlet           Wait-Process                     3.1.0.0      Microsoft.PowerShell.Management
Application      qprocess.exe                     10.0.19...   C:\WINDOWS\system32\qprocess.exe

PS C:\WINDOWS\system32>
```

Get-Command \*Process\*

И вот мы видим список командлетов позволяющих управлять процессами: **Get-Process** – список всех запущенных процессов, **Start-Process** – запустить указанный процесс, **Stop-Process** – остановить указанный процесс, **Wait-Process** – ожидать указанный процесс. Как видно из названий командлетов можно легко понять для чего каждый служит.

Используя командлет **Get-Help** можно получить справку по любому командлету.

```
1 PS C:\WINDOWS\system32> Get-Help Get-Process
2 ИМЯ
3 Get-Process
4 ОПИСАНИЕ
5 Gets the processes that are running on the local computer or a remote computer.
6 СИНТАКСИС
7 Get-Process [[-Name] <System.String[]>] [-ComputerName <System.String[]>]
8 [-FileVersionInfo] [-Module] [<CommonParameters>]
9 Get-Process [-ComputerName <System.String[]>] [-FileVersionInfo] -Id
10 <System.Int32[]> [-Module] [<CommonParameters>]
11 Get-Process [-ComputerName <System.String[]>] [-FileVersionInfo] -InputObject
12 <System.Diagnostics.Process[]> [-Module] [<CommonParameters>]
13 Get-Process -Id <System.Int32[]> -IncludeUserName [<CommonParameters>]
14 Get-Process [[-Name] <System.String[]>] -IncludeUserName
15 [<CommonParameters>]
16 Get-Process -IncludeUserName -InputObject <System.Diagnostics.Process[]>
17 [<CommonParameters>]
18 ОПИСАНИЕ
19 The `Get-Process` cmdlet gets the processes on a local or remote computer.
```

17 Without parameters, this cmdlet gets all of the processes on the local computer.  
You can also specify a particular process b

18 y process name or process ID (PID) or pass a process object through the pipeline  
19 to this cmdlet.

20 By default, this cmdlet returns a process object that has detailed information  
about the process and supports methods that l

21 et you start and stop the process. You can also use the parameters of the `Get-  
22 Process` cmdlet to get file version informati

23 on for the program that runs in the process and to get the modules that the  
process loaded.

24 ССЫЛКИ ПО ТЕМЕ

25 Online Version:  
26 [https://docs.microsoft.com/powershell/module/microsoft.powershell.management/get-  
process?view=powershell-5.1  
&WT.mc\\_id=ps-gethelp](https://docs.microsoft.com/powershell/module/microsoft.powershell.management/get-process?view=powershell-5.1&WT.mc_id=ps-gethelp)

27 Debug-Process

28 Get-Process

29 Start-Process

30 Stop-Process

31 Wait-Process

32 ЗАМЕЧАНИЯ

33 Для просмотра примеров введите: "get-help Get-Process -examples".

34 Для получения дополнительных сведений введите: "get-help Get-Process -detailed".

35 Для получения технических сведений введите: "get-help Get-Process -full".

36 Для получения справки в Интернете введите: "get-help Get-Process -online"

37

38

39

40

41

42

43

44

45

46

47

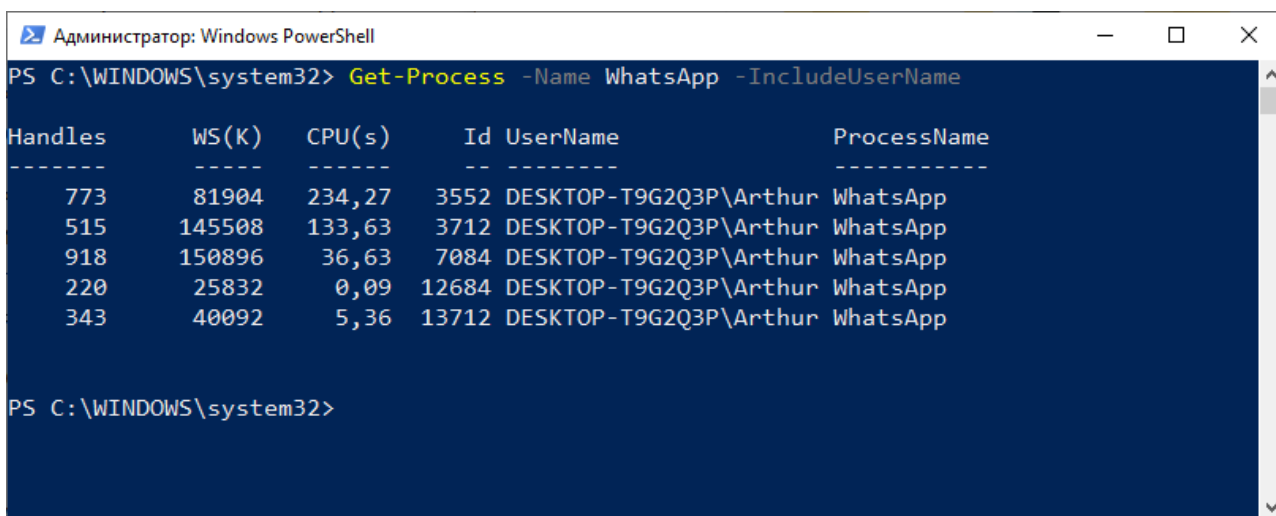
48

49

При использовании командлета есть возможность указать несколько параметров при запуске. Первый параметр можно не называть. Выше я писал **Get-Command \*Process\*** где указал поиск командлетов со словом Process. Однако я не написал параметр **-Name**, хотя именно его и указал. Если полностью то поиск по имени выглядит так: **Get-Command -Name \*Process\***. В случае отсутствия непосредственного указания **-Name** Powershell принимает в качестве имени введенное слово **\*Process\***.

Давайте выведем список процессов с именем **WhatsApp**

```
1 Get-Process -Name WhatsApp -IncludeUserName
```



```
Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Get-Process -Name WhatsApp -IncludeUserName

Handles      WS(K)      CPU(s)      Id  UserName                      ProcessName
-----
773          81904      234,27      3552 DESKTOP-T9G2Q3P\Arthur        WhatsApp
515          145508     133,63      3712 DESKTOP-T9G2Q3P\Arthur        WhatsApp
918          150896     36,63       7084 DESKTOP-T9G2Q3P\Arthur        WhatsApp
220          25832      0,09        12684 DESKTOP-T9G2Q3P\Arthur        WhatsApp
343          40092      5,36        13712 DESKTOP-T9G2Q3P\Arthur        WhatsApp

PS C:\WINDOWS\system32>
```

Get-Process

Мы вывели все процессы с именем **WhatsApp** и добавили в вывод дополнительный параметр **-IncludeUserName**, что позволило нам увидеть кем запущен процесс.

Алиасы

**Алиасы** в Powershell это по сути более короткие названия командлетов. Т.е. любому командлету можно присвоить свое короткое имя (**alias**). Например алиасом для командлета **Get-Process** является **gps**. Согласитесь куда проще и быстрее написать **gps** чем **Get-Process**.

Список всех alias можно получить используя командлет **Get-Alias**

```
1 PS C:\WINDOWS\system32> get-alias
2 CommandType      Name                      Version
3 Source
4 -----
5 Alias            % -> ForEach-Object
6 Alias            ? -> Where-Object
7 Alias            ac -> Add-Content
8 Alias            asnp -> Add-PSSnapin
9 Alias            cat -> Get-Content
```



9	Alias	cd -> Set-Location	
10	Alias	CFS -> ConvertFrom-String	3.1.0.0
	Microsoft.PowerShell.Utility		
11	Alias	chdir -> Set-Location	
12	Alias	clc -> Clear-Content	
13	Alias	clear -> Clear-Host	
14	Alias	clhy -> Clear-History	
15	Alias	cli -> Clear-Item	
16	Alias	clp -> Clear-ItemProperty	
17	Alias	cls -> Clear-Host	
18	Alias	clv -> Clear-Variable	
19	Alias	cnsn -> Connect-PSSession	
20	Alias	compare -> Compare-Object	
21	Alias	copy -> Copy-Item	
22	Alias	cp -> Copy-Item	
23	Alias	cpi -> Copy-Item	
24	Alias	cpp -> Copy-ItemProperty	
25	Alias	curl -> Invoke-WebRequest	
26	Alias	cvpa -> Convert-Path	
27	Alias	dbp -> Disable-PSBreakpoint	
28	Alias	del -> Remove-Item	
29	Alias	diff -> Compare-Object	
30	Alias	dir -> Get-ChildItem	
31	Alias	dnsn -> Disconnect-PSSession	
32	Alias	ebp -> Enable-PSBreakpoint	
33	Alias	echo -> Write-Output	
34	Alias	epal -> Export-Alias	
35	Alias	epcsv -> Export-Csv	
36	Alias	epsn -> Export-PSSession	
37	Alias	erase -> Remove-Item	
38	Alias	etsn -> Enter-PSSession	
39	Alias	exsn -> Exit-PSSession	
40	Alias	fc -> Format-Custom	
	Alias	fhx -> Format-Hex	3.1.0.0
	Microsoft.PowerShell.Utility		
41	Alias	fl -> Format-List	

42	Alias	foreach -> ForEach-Object	
43	Alias	ft -> Format-Table	
44	Alias	fw -> Format-Wide	
45	Alias	gal -> Get-Alias	
46	Alias	gbp -> Get-PSBreakpoint	
47	Alias	gc -> Get-Content	
48	Alias Microsoft.PowerShell.Management	gcb -> Get-Clipboard	3.1.0.0
49	Alias	gci -> Get-ChildItem	
50	Alias	gcm -> Get-Command	
51	Alias	gcs -> Get-PSCallStack	
52	Alias	gdr -> Get-PSDrive	
53	Alias	ghy -> Get-History	
54	Alias	gi -> Get-Item	
55	Alias Microsoft.PowerShell.Management	gin -> Get-ComputerInfo	3.1.0.0
56	Alias	gjb -> Get-Job	
57	Alias	gl -> Get-Location	
58	Alias	gm -> Get-Member	
59	Alias	gmo -> Get-Module	
60	Alias	gp -> Get-ItemProperty	
61	Alias	gps -> Get-Process	
62	Alias	gpv -> Get-ItemPropertyValue	
63	Alias	group -> Group-Object	
64	Alias	gsn -> Get-PSSession	
65	Alias	gsnp -> Get-PSSnapin	
66	Alias	gsv -> Get-Service	
67	Alias Microsoft.PowerShell.Management	gtz -> Get-TimeZone	3.1.0.0
68	Alias	gu -> Get-Unique	
69	Alias	gv -> Get-Variable	
70	Alias	gwmi -> Get-WmiObject	
71	Alias	h -> Get-History	
72	Alias	history -> Get-History	
73	Alias	icm -> Invoke-Command	
74	Alias	iex -> Invoke-Expression	

75	Alias	ihy -> Invoke-History
76	Alias	ii -> Invoke-Item
77	Alias	ipal -> Import-Alias
78	Alias	ipcsv -> Import-Csv
79	Alias	ipmo -> Import-Module
80	Alias	ipsn -> Import-PSSession
81	Alias	irm -> Invoke-RestMethod
82	Alias	ise -> powershell_ise.exe
83	Alias	iwmi -> Invoke-WmiMethod
84	Alias	iwr -> Invoke-WebRequest
85	Alias	kill -> Stop-Process
86	Alias	lp -> Out-Printer
87	Alias	ls -> Get-ChildItem
88	Alias	man -> help
89	Alias	md -> mkdir
90	Alias	measure -> Measure-Object
91	Alias	mi -> Move-Item
92	Alias	mount -> New-PSDrive
93	Alias	move -> Move-Item
94	Alias	mp -> Move-ItemProperty
95	Alias	mv -> Move-Item
96	Alias	nal -> New-Alias
97	Alias	ndr -> New-PSDrive
98	Alias	ni -> New-Item
99	Alias	nmo -> New-Module
100	Alias	npssc -> New-PSSessionConfigurationFile
101	Alias	nsn -> New-PSSession
102	Alias	nv -> New-Variable
103	Alias	ogv -> Out-GridView
104	Alias	oh -> Out-Host
105	Alias	popd -> Pop-Location
106	Alias	ps -> Get-Process
107	Alias	pushd -> Push-Location
	Alias	pwd -> Get-Location
	Alias	r -> Invoke-History

108	Alias	rbp -> Remove-PSBreakpoint	
109	Alias	rcjb -> Receive-Job	
110	Alias	rdsn -> Receive-PSSession	
111	Alias	rd -> Remove-Item	
112	Alias	rdr -> Remove-PSDrive	
113	Alias	ren -> Rename-Item	
114	Alias	ri -> Remove-Item	
115	Alias	rjb -> Remove-Job	
116	Alias	rm -> Remove-Item	
117	Alias	rmdir -> Remove-Item	
118	Alias	rmo -> Remove-Module	
119	Alias	rni -> Rename-Item	
120	Alias	rnp -> Rename-ItemProperty	
121	Alias	rp -> Remove-ItemProperty	
122	Alias	rsn -> Remove-PSSession	
123	Alias	rsnp -> Remove-PSSnapin	
124	Alias	rujb -> Resume-Job	
125	Alias	rv -> Remove-Variable	
126	Alias	rvpa -> Resolve-Path	
127	Alias	rwmi -> Remove-WmiObject	
128	Alias	sajb -> Start-Job	
129	Alias	sal -> Set-Alias	
130	Alias	saps -> Start-Process	
131	Alias	sasv -> Start-Service	
132	Alias	sbp -> Set-PSBreakpoint	
133	Alias	sc -> Set-Content	
134	Alias	scb -> Set-Clipboard	3.1.0.0
135	Microsoft.PowerShell.Management		
136	Alias	select -> Select-Object	
137	Alias	set -> Set-Variable	
138	Alias	shcm -> Show-Command	
139	Alias	si -> Set-Item	
140	Alias	sl -> Set-Location	
	Alias	sleep -> Start-Sleep	
	Alias	sls -> Select-String	

141	Alias	sort -> Sort-Object	
142	Alias	sp -> Set-ItemProperty	
143	Alias	spjb -> Stop-Job	
144	Alias	spps -> Stop-Process	
145	Alias	spsv -> Stop-Service	
146	Alias	start -> Start-Process	
147	Alias	stz -> Set-TimeZone	3.1.0.0
147	Microsoft.PowerShell.Management		
148	Alias	sujb -> Suspend-Job	
149	Alias	sv -> Set-Variable	
150	Alias	swmi -> Set-WmiInstance	
151	Alias	tee -> Tee-Object	
152	Alias	trcm -> Trace-Command	
153	Alias	type -> Get-Content	
154	Alias	wget -> Invoke-WebRequest	
155	Alias	where -> Where-Object	
156	Alias	wjb -> Wait-Job	
157	Alias	write -> Write-Output	
158			
159			
160			
161			
162			

Как видно из списка для alias использованы аналогичные по значению команды из Linux: **ls**, **man**, **mount**, **md**, **kill** и т.п. Видимо чтобы линуксоиду было по привычнее 😊 Можно создать свой alias используя командлет **New-Alias**

Конвейер

Конвейер используется для передачи выходных данных командлета идущего вначале во входные данные командлета следующего за ним. Ничего непонятно? 😊 Давайте на примерах, так всегда яснее.

Возьмем уже известный нам командлет Get-Process, посмотрим на его вывод

```
Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Get-Process
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
199	14	2660	3860	0,20	1452	0	aesm_service
139	8	1436	2128	0,05	3452	0	agent_ovpnconnect_1584645087568
569	33	19944	27132	30,84	9184	1	ApplicationFrameHost
561	25	87064	83460	151,34	6656	0	audiodg
4078	569	299500	108076	10 837,53	3468	0	avp
1566	102	83712	4072	48,61	5508	1	avpui
547	28	21052	904	0,80	7788	1	Calculator
142	11	1888	1828	0,97	9372	1	CNMNSST2
141	7	1656	1772	0,48	10976	1	CompPkgSrv
96	7	6228	1856	0,05	2068	0	conhost
258	13	6504	6076	20,13	9012	1	conhost

Get-Process

Как по мне многовато лишних столбцов. Мне эта информация не нужна, поэтому я выберу только нужные данные. Для таких целей служит командлет `Select-Object`. Давайте используем его в конвейере.

```
1 Get-Process|Select-Object ID,CPU,ProcessName
```

```
Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Get-Process|Select-Object ID,CPU,ProcessName
```

Id	CPU	ProcessName
1452	0,203125	aesm_service
3452	0,046875	agent_ovpnconnect_1584645087568
9184	30,84375	ApplicationFrameHost
6656	151,421875	audiodg
3468	10840,6875	avp
5508	48,796875	avpui
7788	0,796875	Calculator
9372	0,96875	CNMNSST2
10976	0,484375	CompPkgSrv
2068	0,046875	conhost

Get-Process|Sort-Object

Как вы уже наверно догадались конвейер обозначается знаком `|` и идет сразу следом за командлетом. И так данные по конвейеру можно передавать и дальше другим командлетам. И так я передал выходные данные (список запущенных процессов) на вход командлета `Select-Object`. Который в свою очередь выбрал данные по 3 столбцам **ID**, **CPU**, **ProcessName**. Теперь можно передать эти данные дальше. Например выгрузить в текстовый файл

```
1 Get-Process|Select-Object ID,CPU,ProcessName|Out-File C:\TMP\out.txt
```

Просто не правда ли? У нас конвейер из трех командлетов, на выходе которого получаем текстовый файл со списком запущенных процессов и необходимой информацией по ним.

Структура объектов

В Powershell объекты играют самую важную роль. От типа объекта зависит что именно с ним можно сделать. Узнать тип объекта и вывести список всех его элементов позволяет команда **Get-Member**

## 1 `Get-Process|Get-member`

```
Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Get-Process|Get-Member

TypeName: System.Diagnostics.Process

Name                MemberType          Definition
-----
Handles             AliasProperty      Handles = Handlecount
Name                AliasProperty      Name = ProcessName
NPM                 AliasProperty      NPM = NonpagedSystemMemorySize64
PM                  AliasProperty      PM = PagedMemorySize64
SI                  AliasProperty      SI = SessionId
VM                  AliasProperty      VM = VirtualMemorySize64
WS                  AliasProperty      WS = WorkingSet64
Disposed            Event               System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived   Event               System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(System.O
bject, System.Diagnostics.D...
Exited              Event               System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived  Event               System.Diagnostics.DataReceivedEventHandler OutputDataReceived(System.
Object, System.Diagnostics....
BeginErrorReadLine  Method              void BeginErrorReadLine()
BeginOutputReadLine Method              void BeginOutputReadLine()
CancelErrorRead     Method              void CancelErrorRead()
CancelOutputRead    Method              void CancelOutputRead()
Close               Method              void Close()
CloseMainWindow     Method              bool CloseMainWindow()
CreateObjRef        Method              System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose             Method              void Dispose(), void IDisposable.Dispose()
Equals              Method              bool Equals(System.Object obj)
GetHashCode         Method              int GetHashCode()
GetLifetimeService  Method              System.Object GetLifetimeService()
GetType             Method              type GetType()
InitializeLifetimeService Method          System.Object InitializeLifetimeService()
Kill                Method              void Kill()
Refresh             Method              void Refresh()
Start               Method              bool Start()
ToString            Method              string ToString()
WaitForExit         Method              bool WaitForExit(int milliseconds), void WaitForExit()
WaitForInputIdle    Method              bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName          NoteProperty       string __NounName=Process
BasePriority         Property            int BasePriority {get;}
Container            Property            System.ComponentModel.IContainer Container {get;}
EnableRaisingEvents Property            bool EnableRaisingEvents {get;set;}
ExitCode            Property            int ExitCode {get;}
ExitTime            Property            datetime ExitTime {get;}
Handle              Property            System.IntPtr Handle {get;}
HandleCount         Property            int HandleCount {get;}
HasExited           Property            bool HasExited {get;}
Id                  Property            int Id {get;}
```

Get-Process|get-Member

Вот далеко не полный список элементов командлета **Get-Process**. В данном случае тип данных это **System.Diagnostics.Process**

Давайте посмотрим тип данных у новой переменной

- 1 `$new="Test"`
- 2 `$new|Get -Member`

Администратор: Windows PowerShell		
Type Name: System.String		
Name	MemberType	Definition
Clone	Method	System.Object Clone(), System.Object ICloneable.Clone()
CompareTo	Method	int CompareTo(System.Object value), int CompareTo(string strB), int IComparable.Compare...
Contains	Method	bool Contains(string value)
CopyTo	Method	void CopyTo(int sourceIndex, char[] destination, int destinationIndex, int count)
EndsWith	Method	bool EndsWith(string value), bool EndsWith(string value, System.StringComparison compar...
Equals	Method	bool Equals(System.Object obj), bool Equals(string value), bool Equals(string value, Sy...
GetEnumerator	Method	System.CharEnumerator GetEnumerator(), System.Collections.IEnumerator IEnumerable.GetEn...
GetHashCode	Method	int GetHashCode()
GetType	Method	type GetType()
GetTypeCode	Method	System.TypeCode GetTypeCode(), System.TypeCode IConvertible.GetTypeCode()
IndexOf	Method	int IndexOf(char value), int IndexOf(char value, int startIndex), int IndexOf(string va...
IndexOfAny	Method	int IndexOfAny(char[] anyOf), int IndexOfAny(char[] anyOf, int startIndex), int IndexOf...
Insert	Method	string Insert(int startIndex, string value)
IsNormalized	Method	bool IsNormalized(), bool IsNormalized(System.Text.NormalizationForm normalizationForm)
LastIndexOf	Method	int LastIndexOf(char value), int LastIndexOf(char value, int startIndex), int LastIndex...
LastIndexOfAny	Method	int LastIndexOfAny(char[] anyOf), int LastIndexOfAny(char[] anyOf, int startIndex), int...
Normalize	Method	string Normalize(), string Normalize(System.Text.NormalizationForm normalizationForm)
PadLeft	Method	string PadLeft(int totalWidth), string PadLeft(int totalWidth, char paddingChar)
PadRight	Method	string PadRight(int totalWidth), string PadRight(int totalWidth, char paddingChar)
Remove	Method	string Remove(int startIndex, int count), string Remove(int startIndex)
Replace	Method	string Replace(char oldChar, char newChar), string Replace(string oldValue, string newV...
Split	Method	string[] Split(Params char[] separator), string[] Split(char[] separator, int count), s...
StartsWith	Method	bool StartsWith(string value), bool StartsWith(string value, System.StringComparison co...
Substring	Method	string Substring(int startIndex), string Substring(int startIndex, int length)
ToBoolean	Method	bool IConvertible.ToBoolean(System.IFormatProvider provider)
ToByte	Method	byte IConvertible.ToByte(System.IFormatProvider provider)
ToChar	Method	char IConvertible.ToChar(System.IFormatProvider provider)
ToCharArray	Method	char[] ToCharArray(), char[] ToCharArray(int startIndex, int length)
DateTime	Method	datetime IConvertible.ToDateTime(System.IFormatProvider provider)
ToDecimal	Method	decimal IConvertible.ToDecimal(System.IFormatProvider provider)
ToDouble	Method	double IConvertible.ToDouble(System.IFormatProvider provider)
ToInt16	Method	int16 IConvertible.ToInt16(System.IFormatProvider provider)
ToInt32	Method	int IConvertible.ToInt32(System.IFormatProvider provider)
ToInt64	Method	long IConvertible.ToInt64(System.IFormatProvider provider)
ToLower	Method	string ToLower(), string ToLower(cultureinfo culture)
ToLowerInvariant	Method	string ToLowerInvariant()
ToSByte	Method	sbyte IConvertible.ToSByte(System.IFormatProvider provider)
ToSingle	Method	float IConvertible.ToSingle(System.IFormatProvider provider)
ToString	Method	string ToString(), string ToString(System.IFormatProvider provider), string IConvertibl...
ToType	Method	System.Object IConvertible.ToType(type conversionType, System.IFormatProvider provider)
ToUInt16	Method	uint16 IConvertible.ToUInt16(System.IFormatProvider provider)
ToUInt32	Method	uint32 IConvertible.ToUInt32(System.IFormatProvider provider)
ToUInt64	Method	uint64 IConvertible.ToUInt64(System.IFormatProvider provider)
ToUpper	Method	string ToUpper(), string ToUpper(cultureinfo culture)
ToUpperInvariant	Method	string ToUpperInvariant()
Trim	Method	string Trim(Params char[] trimChars), string Trim()
TrimEnd	Method	string TrimEnd(Params char[] trimChars)

Get-Member String

В данном случае тип данных **System.String** т.е. строка. Что вполне логично. А теперь посмотрите что можно сделать с этой строкой с учетом указанных выше параметров.

```

Администратор: Windows PowerShell
PS C:\WINDOWS\system32> ($new).ToUpper()
TEST
PS C:\WINDOWS\system32> ($new).ToLower()
test
PS C:\WINDOWS\system32> ($new).Length
4
PS C:\WINDOWS\system32>

```

Get-Member methods

Как видно на картинке выше мы заключаем нашу тестовую переменную **\$new** в скобки и после них пишем точку и указываем метод. В примере я использовал три метода:

- **ToUpper** – перевод всех букв в строке в верхний регистр
- **ToLower** – перевод всех букв в строке в нижний регистр
- **Length** – подсчитать количество символов в строке



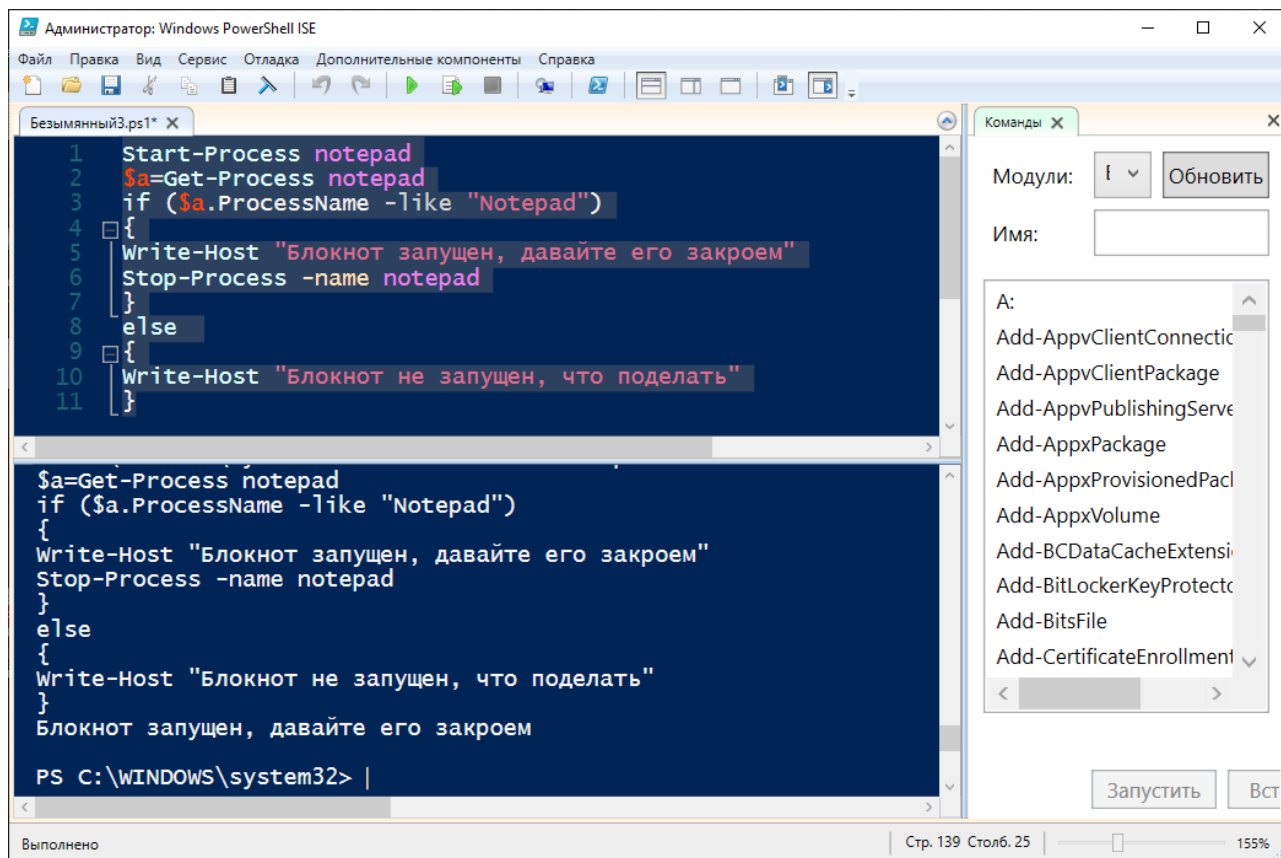
Это всего лишь небольшой пример что можно сделать с параметрами объекта. Чаше используйте **Get-Member** и вы откроете для себя безграничные возможности манипуляции над объектами.

Скрипты Powershell

В самом начале статьи указал на встроенный инструмент **Powershell ISE**. Давайте запустим его и создадим свой первый скрипт. Кстати скрипты сохраняются в файлах с расширением **ps1**

Скрипт будет запускать блокнот, далее выполняется проверка если блокнот запущен выводится сообщение об этом и после блокнот закрывается. Если блокнот не запущен то выводится соответствующее сообщение об этом. На самом деле блокнот будет всегда запущен, т.к. мы вначале скрипта написали **Start-Process notepad**

```
1  Start-Process notepad
2  $a=Get-Process notepad
3  if ($a.ProcessName -like "Notepad")
4  {
5      Write-Host "Блокнот запущен, давайте его закроем"
6      Stop-Process -name notepad
7  }
8  else
9  {
10     Write-Host "Блокнот не запущен, что поделать"
11 }
```



PowerShell скрипты

В этом скрипте я использовал цикл **if else**. О циклах будет подробнее в следующей статье. Итак давайте сохраним скрипт и выполним его.

В ответ мы получим такую ошибку:

```

1 Невозможно загрузить файл, так как выполнение сценариев отключено в этой
  системе. Для получения дополнительных сведений см.
2 about_Execution_Policies по адресу https://go.microsoft.com/fwlink/?LinkID=1351
3 70.
4 + CategoryInfo          : Ошибка безопасности: (:) [], ParentContainsError
5 RecordException
6 + FullyQualifiedErrorId : UnauthorizedAccess

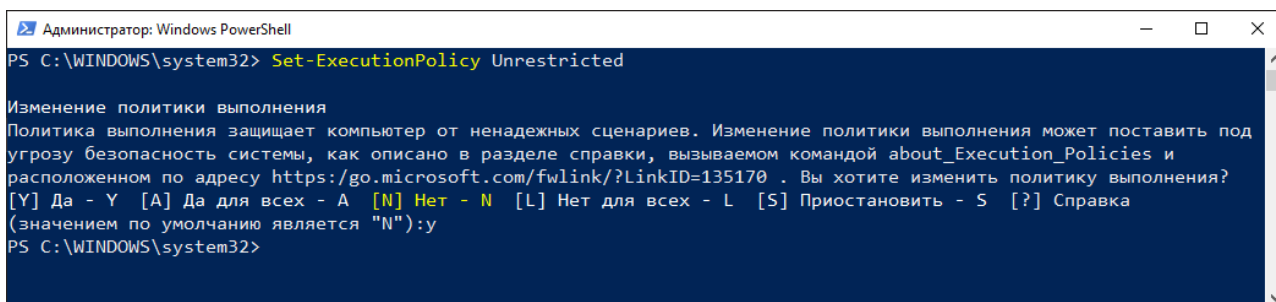
```

Все верно, изначально в Windows запрещено выполнять скрипты PowerShell. Это сделано для повышения безопасности системы. Для включения возможности запуска скриптов PowerShell необходимо запустить PowerShell от Администратора и ввести командлет **Set-ExecutionPolicy** с одним из параметров:

- **Restricted** – политика по умолчанию. Выполнение всех скриптов запрещено
- **RemoteSigned** – разрешено запускать собственные скрипты и подписанные доверенным разработчиком
- **AllSigned** – разрешено запускать скрипты, подписанные доверенным разработчиком. Каждый раз перед запуском такого скрипта PowerShell будет запрашивать подтверждение
- **Unrestricted** – в системе разрешается запускать любые скрипты

Если вы полностью уверены в запускаемых скриптах можете поставить Unrestricted. Давайте так и сделаем

#### 1 Set-ExecutionPolicy -Unrestricted



```
Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Set-ExecutionPolicy Unrestricted

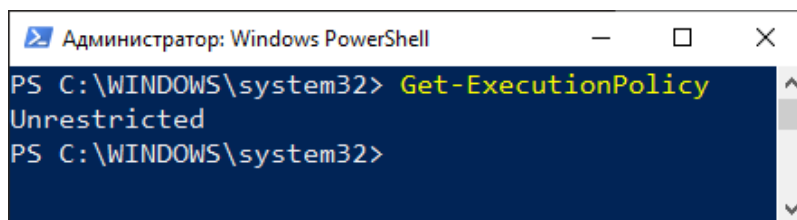
Изменение политики выполнения
Политика выполнения защищает компьютер от ненадежных сценариев. Изменение политики выполнения может поставить под угрозу безопасность системы, как описано в разделе справки, вызываемом командой about_Execution_Policies и расположенном по адресу https://go.microsoft.com/fwlink/?LinkID=135170 . Вы хотите изменить политику выполнения?
[Y] Да - Y [A] Да для всех - A [N] Нет - N [L] Нет для всех - L [S] Приостановить - S [?] Справка
(значением по умолчанию является "N"):y
PS C:\WINDOWS\system32>
```

Set-ExecutionPolicy

Будет предупреждение по безопасности, соглашаемся нажав Y

Можем посмотреть текущую настройку политики безопасности при помощи командлета **Get-ExecutionPolicy**

#### 1 Get-ExecutionPolicy



```
Администратор: Windows PowerShell
PS C:\WINDOWS\system32> Get-ExecutionPolicy
Unrestricted
PS C:\WINDOWS\system32>
```

Get-ExecutionPolicy

В данной статье мы рассмотрели основы чтобы подготовиться писать скрипты Powershell. В следующих статьях мы более подробно изучим циклы, массивы, функции, работу со строками и много другое. Кстати вот раздел посвященный Powershell. Там много всего интересного 😊

Рекомендую к прочтению:

- Переменные
- Операторы сравнения
- Операторы условий
- Циклы