

Powershell

 **admin**

Массивы – одна из основных функций большинства языков программирования. Массив состоит из коллекции элементов. Выполнив итерацию (проход) по массиву можно получить доступ к любому элементу коллекции. Массивы Powershell можно создать несколькими способами. Рассмотрим их подробнее.

Создание массива

Создать массив можно при помощи конструкции **@()**

```
1 $massiv=@()
```

```
Windows PowerShell
PS C:\> $massiv=@()
PS C:\> $massiv
PS C:\> $massiv.GetType()

IsPublic IsSerial Name BaseType
-----
True     True     Object[] System.Array

PS C:\> |
```

Я создал переменную **\$massiv** и добавил в нее пустой массив. Как видно на картинке переменная имеет тип массив (**System.Array**). Добавлять элементы в строковый массив можно через запятую.

```
1 $massiv=@"odin","dva","tri"
```

Возможен и другой тип записи, каждый элемент с новой строки

```
1 $massiv=@"Petya"
2 "Vasya"
3 "Masha")
```

Массив можно создать просто объявив переменную с перечисленным списком значений

```
1 $newmassiv="volga","lada","uaz"
```

```
Windows PowerShell
PS C:\> $newmassiv="volga","lada","uaz"
PS C:\> $newmassiv.GetType()

IsPublic IsSerial Name BaseType
-----
True     True     Object[] System.Array

PS C:\> |
```

В массиве допускается использование данных разных типов. Например можно записать строку и число

```
1 $newmassiv="volga",3,"uaz"
```

```
PS C:\> $newmassiv[0].GetType()

IsPublic IsSerial Name BaseType
-----
True     True     String System.Object

PS C:\> $newmassiv[1].GetType()

IsPublic IsSerial Name BaseType
-----
True     True     Int32   System.ValueType

PS C:\>
```

Работа с элементами массива

Доступ к элементу массива можно получить с помощью квадратный скобок [] указав внутри номер элемента. Номера элементов массива начинаются с 0.

```
1 $test="raz","dva","tri"
```

Вывод элементов

Доступ к первому элементу массива можно получить так:

```
1 $test[0]
2 raz
```

Удобство в том, что доступ можно получить сразу к нескольким элементам, указав их через запятую.

```
1 $test[0,1]
2 raz
3 dva
```

Либо указав начальный и конечный элемент используя оператор ..

```
1 $test[0..2]
2 raz
3 dva
4 tri
```

Вывод элементов в обратном порядке также возможен

```
1 $test[2..0]
2 tri
3 dva
4 raz
```

Когда необходимо вывести последний элемент с конца массива можно использовать **-1**

```
1 $test[-1]
2 tri
```

Подсчитать количество элементов в массиве можно с помощью свойства **Count**

```
1 $test.Count
2 3
```

Используя подсчет элементов в массиве **Count** можно, например, контролировать количество запущенных процессов. Рассмотрим пример в котором подсчитаем количество запущенных процессов *firefox*. Если процессов более **10**, будем выводить сообщение.

```
1 $gp=Get-Process -Name firefox
2 $gp.Count
3 if ($gp.Count -ge 10) {Write-Host "Многовато открыто вкладок, закрой парочку"}
```

```
Windows PowerShell
PS C:\> $gp=Get-Process -Name firefox
PS C:\> $gp.Count
13
PS C:\> if ($gp.Count -ge 10) {Write-Host "Многовато открыто вкладок, закрой парочку"}
Многовато открыто вкладок, закрой парочку
PS C:\>
|
```

Обновление элементов

Обновить значение элементов в массиве можно используя его номер и указав новое значение взамен старого

- 1 `$test[1]="chetire"`
- 2 `$test[2]="pyat"`

Причем таким способом мы именно обновляем уже существующие элементы в массиве. Добавить новый элемент не получится. Если ввести команду `$test[3]="shest"` появится ошибка **“Индекс находился вне границ массива.”**

Обновление значений элементов используя циклы

Как правило массив является коллекцией строковых или целочисленных значений. Обычно, в цикле используется переменная, которая содержит копию значения. При обновлении этой переменной исходное значение в массиве не обновляется. С решением данной проблемы поможет цикл **for**.

- 1 `for ($a = 0; $a -lt $massiv.count; $a++)`
- 2 `{`
- 3 `$massiv[$a] = "Item: [{0}]" -f $massiv[$a]`
- 4 `}`

В данном примере принимается значение по индексу `$a`, далее происходит изменение значения и затем этот же индекс используется для внесения значения обратно.

Математические операции с элементами

С элементами массива можно выполнять математические операции. Давайте создадим массив чисел

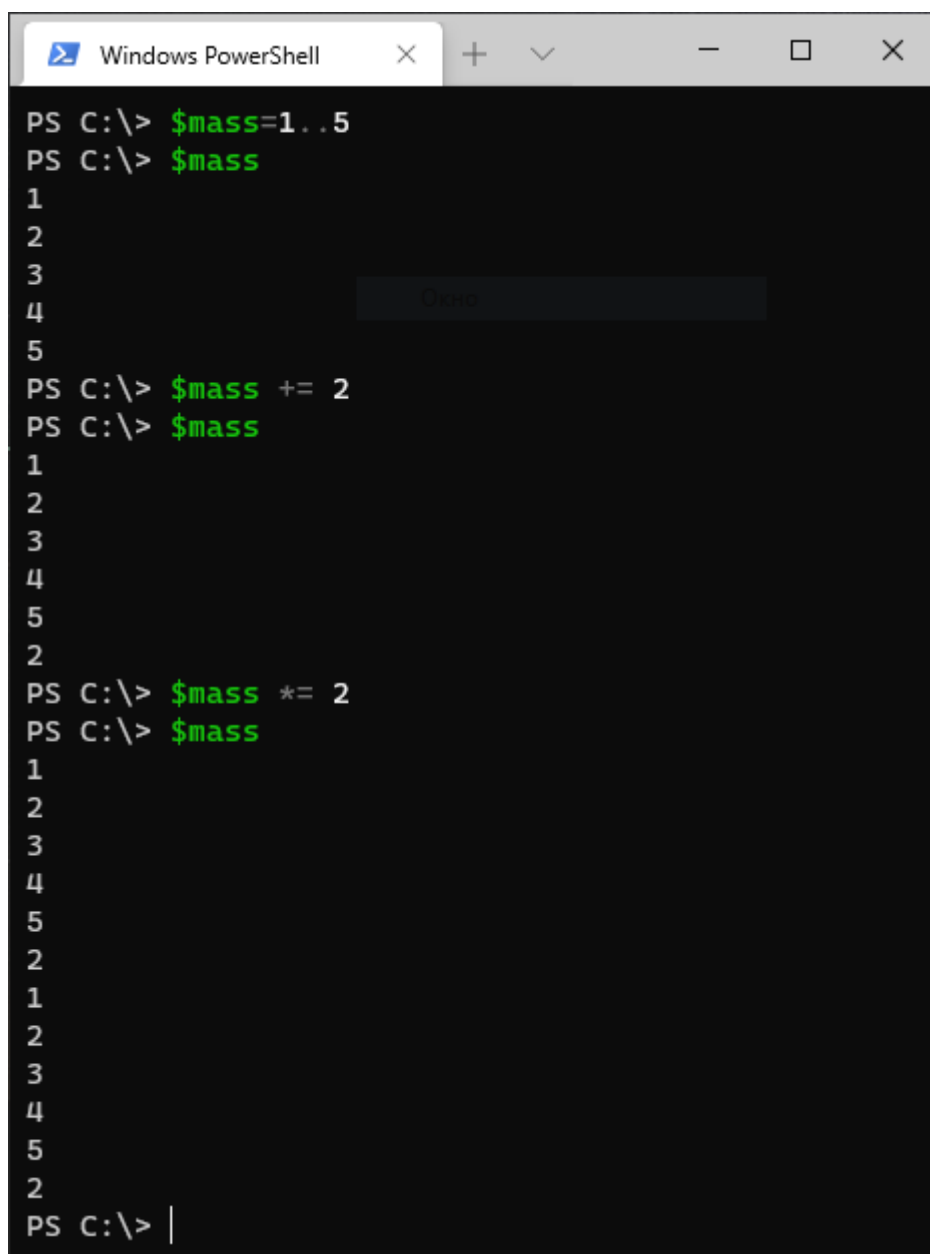
- 1 `$mass=1..5`

Затем добавим к нему элемент

```
1 $mass += 2
```

И уже обновленный массив умножим на 2

```
1 $mass *= 2
```

A screenshot of a Windows PowerShell terminal window. The window title is "Windows PowerShell". The terminal shows the following commands and output:
PS C:\> \$mass=1..5
PS C:\> \$mass
1
2
3
4
5
PS C:\> \$mass += 2
PS C:\> \$mass
1
2
3
4
5
2
PS C:\> \$mass *= 2
PS C:\> \$mass
1
2
3
4
5
2
1
2
3
4
5
2
PS C:\> |
The output shows the array elements after each operation. After the first operation, the array is 1, 2, 3, 4, 5. After adding 2, it becomes 1, 2, 3, 4, 5, 2. After multiplying by 2, it becomes 1, 2, 3, 4, 5, 2, 1, 2, 3, 4, 5, 2.

```
Windows PowerShell
PS C:\> $mass=1..5
PS C:\> $mass
1
2
3
4
5
PS C:\> $mass += 2
PS C:\> $mass
1
2
3
4
5
2
PS C:\> $mass *= 2
PS C:\> $mass
1
2
3
4
5
2
1
2
3
4
5
2
PS C:\> |
```

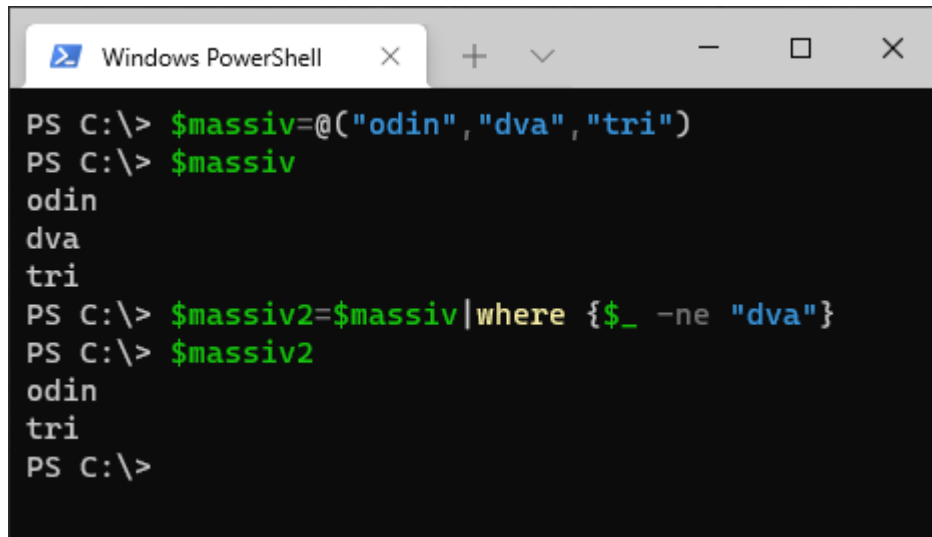
Арифметические операции

Удаление элементов

В Powershell массивы являются неизменяемыми и удалить элемент напрямую не получится. Варианты решения этого вопроса следующие:

Вариант №1: создать новый массив без элемента который хотим удалить.

```
1 $massiv=@("odin","dva","tri")
2 $massiv
3 $massiv2=$massiv|where {$_ -ne "dva"}
4 $massiv2
```



```
Windows PowerShell
PS C:\> $massiv=@("odin","dva","tri")
PS C:\> $massiv
odin
dva
tri
PS C:\> $massiv2=$massiv|where {$_ -ne "dva"}
PS C:\> $massiv2
odin
tri
PS C:\>
```

Создание нового массива из старого

Переменной **\$massiv** было присвоено **3** значения, затем была создана переменная **\$massiv2** в которую записали **\$massiv** с исключением второго элемента.

Вариант №2: преобразовать массив к изменяемому типу **ArrayList** с возможностью удаления элементов

```
1 $massiv=@("odin","dva","tri")
2 $massiv
3 $massiv=[System.Collections.ArrayList]$massiv
4 $massiv.RemoveAt($massiv.Count-2)
5 $massiv
```

```
Windows PowerShell
PS C:\> $massiv=@("odin","dva","tri")
PS C:\> $massiv
odin
dva
tri
PS C:\> $massiv=[System.Collections.ArrayList]$massiv
PS C:\> $massiv.RemoveAt($massiv.Count-2)
PS C:\> $massiv
odin
tri
PS C:\>
```

Массив типа ArrayList

После преобразования массива в ArrayList мы успешно удалили значение *dva*

Массивы и циклы

Массивы регулярно используются в циклах. Рассмотрим простейший пример массива данных с передачей его значений по конвейеру циклу **ForEach-Object** (алиас **foreach**)

```
1 "newadmin.ru","vipadm.ru"|foreach {ping $_}
```



```
Windows PowerShell
PS C:\> "newadmin.ru","vipadm.ru"|foreach {ping $_}

Обмен пакетами с newadmin.ru [31.31.196.193] с 32 байтами данных:
Ответ от 31.31.196.193: число байт=32 время=6мс TTL=55
Ответ от 31.31.196.193: число байт=32 время=7мс TTL=55
Ответ от 31.31.196.193: число байт=32 время=7мс TTL=55
Ответ от 31.31.196.193: число байт=32 время=7мс TTL=55

Статистика Ping для 31.31.196.193:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 6мсек, Максимальное = 7 мсек, Среднее = 6 мсек

Обмен пакетами с vipadm.ru [31.31.196.2] с 32 байтами данных:
Ответ от 31.31.196.2: число байт=32 время=7мс TTL=54
Ответ от 31.31.196.2: число байт=32 время=6мс TTL=54
Ответ от 31.31.196.2: число байт=32 время=6мс TTL=54
Ответ от 31.31.196.2: число байт=32 время=8мс TTL=54

Статистика Ping для 31.31.196.2:
    Пакетов: отправлено = 4, получено = 4, потеряно = 0
    (0% потерь)
Приблизительное время приема-передачи в мс:
    Минимальное = 6мсек, Максимальное = 8 мсек, Среднее = 6 мсек
PS C:\> |
```

Массив и цикл ForeEach-Object

В данном случае мы создали массив из двух переменных с названиями сайтов и далее передали эти переменные по конвейеру циклу ForEach-Object. В теле цикла происходит поочередное обращение команды **ping** к каждому элементу массива.

Цикл ForEach-Object работает только с конвейером, для всех остальных случаев есть цикл **ForEach**. Рассмотрим на его примере работу с массивами.

- 1 `$serv="newadmin.ru","vipadm.ru"`
- 2 `foreach ($s in $serv)`
- 3 `{Test-NetConnection -ComputerName $s -Port 80}`

```
Windows PowerShell
PS C:\> $serv="newadmin.ru","vipadm.ru"
PS C:\> foreach ($s in $serv){Test-NetConnection -ComputerName $s -Port 80}

ComputerName      : newadmin.ru
RemoteAddress     : 31.31.196.193
RemotePort        : 80
InterfaceAlias    : hm
SourceAddress     : 172.16.2.33
TcpTestSucceeded  : True

ComputerName      : vipadm.ru
RemoteAddress     : 31.31.196.2
RemotePort        : 80
InterfaceAlias    : hm
SourceAddress     : 172.16.2.33
TcpTestSucceeded  : True

PS C:\> |
```

Массив и цикл ForEach

В этом простом сценарии мы записали в переменную **\$serv** названия сайтов, далее циклом **foreach** прошли по массиву **\$serv** поочередно используя каждый элемент (через переменную **\$s**). Командлетом **Test-NetConnection** проверили доступность порта **80** для каждого из сайтов. Очень удобно если нужно мониторить большое количество площадок в интернете, серверов и т.д.

Массивы и операторы

Использование операторов расширяет удобство использования массивов данных. Перечислю основные операторы с примерам их работы.

Split

Группирует элементы в две разные коллекции. Рассмотрим пример со списком запущенных процессов, среди них выделим процесс **firefox**.

```
1 $firefox, $other = (Get-Process).Where({$_.ProcessName -eq 'firefox'},
   'Split')
2 $firefox
```

```
Windows PowerShell
PS C:\> $firefox, $other = (Get-Process).Where({$_ .ProcessName -eq 'firefox'}, 'Split')
PS C:\> $firefox
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
486	25	33796	17816	4,77	1444	1	firefox
465	48	85552	71632	12,81	3004	1	firefox
422	34	47980	41376	4,05	5828	1	firefox
431	36	70068	50356	5,50	6032	1	firefox
444	58	111056	84888	622,23	7020	1	firefox
2338	194	379592	346696	1 837,59	7436	1	firefox
432	38	71896	83044	47,22	7656	1	firefox
434	49	115096	88376	10,67	7852	1	firefox
439	38	72104	96732	9,73	7880	1	firefox

Массив с процессами firefox

Мы объявили сразу две переменные **\$firefox** и **\$other**. Запросив список запущенных процессов происходит фильтрация процессов с именем **firefox**. Отфильтрованные процессы попадают в переменную **\$firefox**, остальные процессы попадают в переменную **\$other**.

```
Windows PowerShell
PS C:\> $other
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
110	8	2036	1956		1888	0	amdlogsr
206	13	2556	1452	0,27	8548	1	amdow
581	23	5960	6212	4,88	260	1	AMDRSServ
590	35	31604	22896	12,94	10372	1	ApplicationFrameHost
127	8	1572	312		3964	0	armsvc
448	16	4168	7104		6616	1	atieclxx
191	9	1612	1976		1896	0	atiesrxx
216	14	9608	11804	1 914,55	12536	0	audiodg
110	7	1380	1112		17064	0	AUEPLauncher
382	18	7520	12856		17144	0	AUEPMaster

Массив с остальными процессами

First

При формировании вывода можно ограничиться только небольшим количеством переменных. Оператор **First** позволяет вывести определенное количество элементов в начале списка.

```
1 $other.Where({$_ .CPU -gt "1000"}, 'First', 3)
```

```
Windows PowerShell
7118 130 254012 196416 3 223,95 12208 1 OUTLOOK
1166 125 220460 172056 1 133,48 12280 1 Skype

PS C:\> $other.Where({$_ .CPU -gt "1000"}, 'First', 3)

Handles NPM(K) PM(K) WS(K) CPU(s) Id SI ProcessName
-----
216 14 9608 11804 1 920,81 12536 0 audiodg
3852 791 154040 134600 1 412,61 7460 1 explorer
534 52 48604 51184 1 498,34 2080 1 MobaXterm_Personal_21.4

PS C:\> |
```

Вывод первых 3 элементов

Отфильтровав массив **\$other** по параметру **CPU** более **1000** запрошен вывод только первых **3** элементов массива.

Last

Принцип работы **Last** схож с оператором First. Только если First выводит определенное количество элементов вначале массива, то Last выводит необходимое количество с конца списка.

- 1 `$service=(Get-Service).Where({$_ .Name -like 'Win*'}, 'Last', 3)`
- 2 `$service`

```
Windows PowerShell
PS C:\> $service=(Get-Service).Where({$_ .Name -like 'Win*'}, 'Last', 3)
PS C:\> $service

Status Name DisplayName
-----
Running WinHttpAutoProx... Служба автоматического обнаружения ...
Running Winmgmt Инструментарий управления Windows
Running WinRM Служба удаленного управления Window...

PS C:\> |
```

Вывод последних 3 элементов

В данном примере переменная **\$service** получила значения всех сервисов с именем **Win** в начале. Вывод был ограничен **3** значениями с конца списка с помощью оператора Last.

Часто используемые операторы я перечислил, но их гораздо больше. Описание с примерам уже рассмотрено в моей предыдущей [статье](#).

Больше интересных статей о Powershell читайте на нашем сайте.