МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Российский государственный университет им. А.Н. Косыгина   
(Технологии. Дизайн. Искусство)»**

Институт ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И ЦИФРОВЫЕ ТРАНСФОРМАЦИИ

**Отчет по лабораторной работе № 3**

**по дисциплине «Операционные системы»**

**Тема: «Скрипты PowerShell»**

Выполнил: Сидоров Д. С., группа ИТС-123

Проверил: к.т.н., доц. Семёнов А. А.

Москва, 2025г

**Скрипты PowerShell**

1. Удаление ненужных приложений. С помощью следующего скрипта можно удалить предустановленные приложения:

get-appxpackage -name \*APPNAME\* | remove-appxpackage

\*APPNAME\* - название ненужного приложения.

2. Управление процессами.

Воспользуйтесь PowerShell для борьбы с медленными процессами:

а) выведите все службы: Get-Service

б) получите информацию о конкретной службе с кодовым именем \*NAME\*: Get-Service \*NAME\*

в) создайте скрипт, который закрывает процессы с повышенным потреблением ресурсов:

Stop-Service -Name \*имя\_процесса\*

Stop-Service -Name \*имя\_процесса\*

3. Переименование группы файлов.

Решите проблему однотипных файлов с помощью скрипта группового переименования. Укажите путь, расширение и выполните замены в строке.

$path = «$comp\desktop\journey\russia»

$filter = ‘\*.jpg’

get-childitem -path $path -filter $filter | rename-item -newname {$\_.name -replace ‘HGNMD’,’RUSSIA’}

4. Поиск файлов.

Используйте PowerShell для поиска файлов в директории:

Get-Childitem C:\Windows\\*.log

Для более сложного поиска в подпапках:

Get-ChildItem C:\Windows\\* -Include \*.log -Recurse -Force

5. Скрипт, создающий новую папку:

$folderPath = «C:\Path\To\NewFolder»

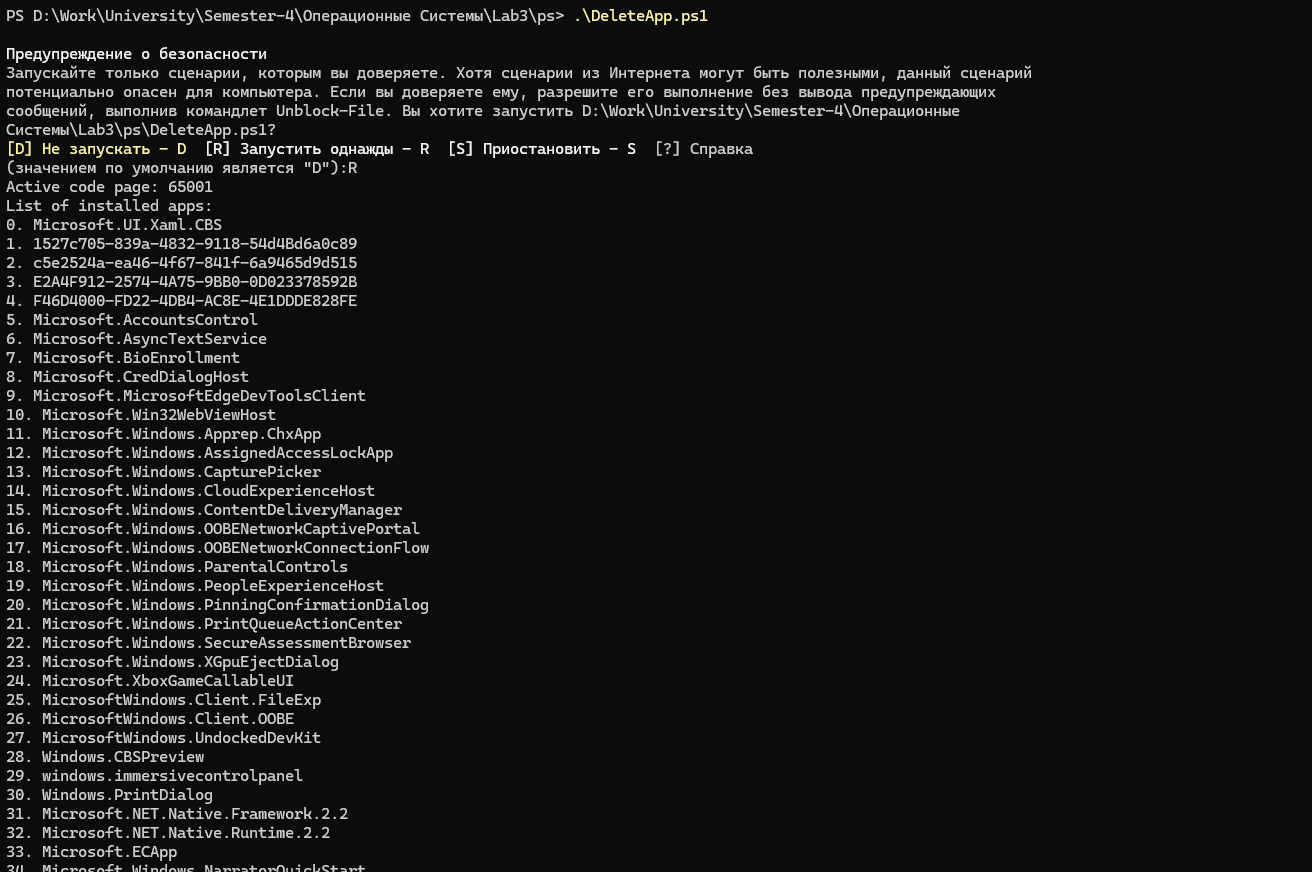
New-Item -ItemType Directory -Path $folderPath

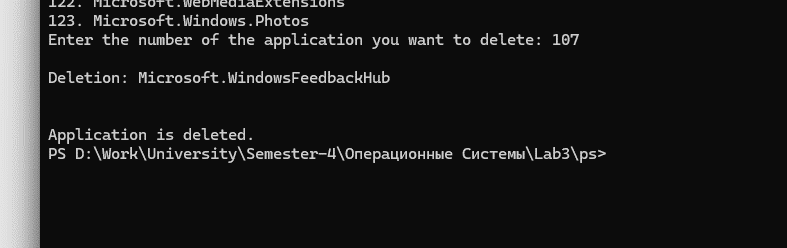
6. Скрипт, удаляющий все файлы старше 7 дней в папке:

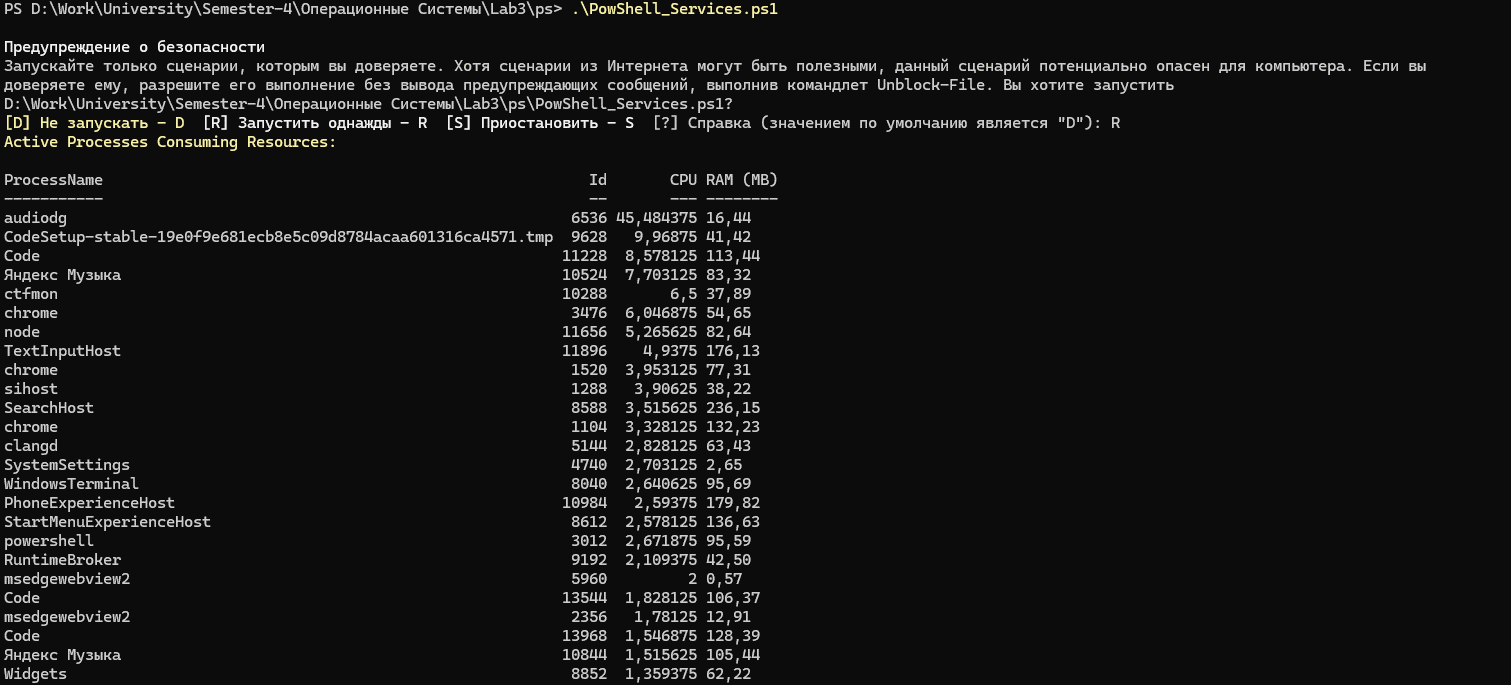
$folderPath = «C:\Path\To\Folder»

$limitDate = (Get-Date).AddDays(-7)

Get-ChildItem $folderPath | Where-Object { $\_.LastWriteTime -lt $limitDate } | Remove-Item

*  
Рисунок 1. Работа скрипта из п. 1.*

 *Рисунок 2. Удаление приложения из п1.*

 *Рисунок 3. Работа скрипта из п. 2.*

**Листинг кода**

**Скрипт п.1.**

[Console]::OutputEncoding = [System.Text.Encoding]::UTF8

chcp 65001

$apps = Get-AppxPackage | Select-Object Name, PackageFullName

# Показываем список приложений с нумерацией

Write-Host "List of installed apps:"

for ($i = 0; $i -lt $apps.Count; $i++) {

Write-Host "$i. $($apps[$i].Name)"

}

[int]$choice = Read-Host "Enter the number of the application you want to delete"

# Проверка корректности ввода

if ($choice -ge 0 -and $choice -lt $apps.Count) {

$selectedApp = $apps[$choice]

Write-Host "`nDeletion: $($selectedApp.Name)`n"

# Удаление выбранного приложения

Get-AppxPackage -Name $selectedApp.Name | Remove-AppxPackage

Write-Host "`nApplication is deleted."

} else {

Write-Host "`nWrong enter. Closing of script."

}

**Скрипт п.2.**

# Thresholds

$cpuThreshold = 10 # Минимальная загрузка CPU для отображения

$ramThreshold = 50MB # Минимальная память для отображения

# Получение всех процессов без системных

$allProcesses = Get-Process | Where-Object {

$\_.ProcessName -notin @('Idle', 'System', 'smss')

}

# Фильтрация по владельцу с обработкой ошибок

$filteredProcesses = @()

foreach ($proc in $allProcesses) {

try {

$owner = (Get-WmiObject Win32\_Process -Filter "ProcessId = $($proc.Id)" | Select-Object -ExpandProperty GetOwner -ErrorAction Stop)

# Проверка владельца, исключение системных пользователей

if ($owner -and $owner.User -notin @('SYSTEM', 'LOCAL SERVICE', 'NETWORK SERVICE')) {

$filteredProcesses += $proc

}

} catch {

# Игнорирование процессов, к которым нет доступа

continue

}

}

# Фильтр по потреблению ресурсов

$filteredProcesses = $allProcesses | Where-Object {

$\_.CPU -le $cpuThreshold -or $\_.WorkingSet64 -le $ramThreshold

} | Sort-Object CPU -Descending

# Вывод информации

if ($filteredProcesses) {

Write-Host "Active Processes Consuming Resources:" -ForegroundColor Yellow

$filteredProcesses | Select-Object ProcessName, Id, CPU,

@{Name="RAM (MB)"; Expression={($\_.WorkingSet64 / 1MB).ToString("0.00")}} |

Format-Table -AutoSize

} else {

Write-Host "No resource-consuming processes found." -ForegroundColor Cyan

Exit

}

# Выбор процессов для завершения

Write-Host "`nEnter the ID(s) of the processes to terminate (comma or space separated):" -ForegroundColor Magenta

$userInput = Read-Host "Process ID(s)"

# Проверка и завершение процессов

if ($userInput) {

# Разделение строк на массив ID

$processIds = $userInput -split '[, ]+' | ForEach-Object { $\_.Trim() } | Where-Object { $\_ -match '^\d+$' }

if ($processIds) {

foreach ($id in $processIds) {

try {

Stop-Process -Id $id -Force -ErrorAction Stop

Write-Host "Process with ID $id terminated." -ForegroundColor Green

} catch {

Write-Host "Failed to terminate process with ID $id." -ForegroundColor Red

}

}

} else {

Write-Host "No valid process IDs entered." -ForegroundColor Red

}

} else {

Write-Host "No processes selected for termination." -ForegroundColor Cyan

}

Write-Host "`nPress any key to exit..." -ForegroundColor Magenta

$null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")

**Скрипт п.3.**

# Requesting user input

$folderPath = Read-Host "Enter the folder path"

$extension = Read-Host "Enter the file extension (e.g., txt, jpg, ps1)"

$oldPart = Read-Host "Enter the part of the name you want to replace"

$newPart = Read-Host "Enter the new part of the name"

# Check if the folder exists

if (Test-Path $folderPath) {

# Search for files with the specified extension

$files = Get-ChildItem -Path $folderPath -Filter "\*.$extension" -File

if ($files.Count -eq 0) {

Write-Host "No files with the extension '$extension' were found."

}

else {

foreach ($file in $files) {

# Check if the file name contains the old part

if ($file.Name -match $oldPart) {

$newName = $file.Name -replace $oldPart, $newPart

$newPath = Join-Path -Path $folderPath -ChildPath $newName

# Rename the file

Rename-Item -Path $file.FullName -NewName $newName

Write-Host "File '$($file.Name)' renamed to '$newName'"

}

}

}

}

else {

Write-Host "The folder '$folderPath' does not exist."

}

**Скрипт п.4.**

$folderPath = Read-Host "Enter the folder path"

$fileName = Read-Host "Enter the file name or part of it (e.g., report.txt or log)"

# Check if the folder exists

if (Test-Path $folderPath) {

# Search for the file (including subfolders)

$files = Get-ChildItem -Path $folderPath -Recurse -File | Where-Object { $\_.Name -like "\*$fileName\*" }

if ($files) {

Write-Host "`nFound files:"

foreach ($file in $files) {

Write-Host "$($file.Name)"

}

}

else {

Write-Host "No files matching '$fileName' were found in '$folderPath'."

}

}

else {

Write-Host "The folder '$folderPath' does not exist."

}

**Скрипт п.5.**

if ($args.Count -eq 0) {

Write-Host "Please provide the folder name as an argument."

Write-Host "Usage: .\CreateFolder.ps1 <FolderName> [<Path>]"

exit

}

$newFolderName = $args[0]

if ($args.Count -eq 2) {

$folderPath = $args[1]

} else {

$folderPath = Get-Location

}

# Формируем полный путь к новой папке

$newFolderPath = Join-Path -Path $folderPath -ChildPath $newFolderName

if (Test-Path $newFolderPath) {

Write-Host "The folder '$newFolderPath' already exists."

}

else {

New-Item -ItemType Directory -Path $newFolderPath | Out-Null

Write-Host "Folder created: $newFolderName"

}

**Скрипт п.6.**

$folderPath = Read-Host "Enter the folder path"

if (Test-Path $folderPath) {

$files = Get-ChildItem -Path $folderPath -File

$currentDate = Get-Date

# Перебираем файлы

foreach ($file in $files) {

if (($currentDate - $file.LastWriteTime).Days -gt 7) {

Remove-Item -Path $file.FullName -Force

Write-Host "File '$($file.Name)' deleted."

}

}

Write-Host "Finished checking files."

}

else {

Write-Host "The folder '$folderPath' does not exist."

}

**Вывод:** В ходе лабораторной работы были изучены основные возможности PowerShell для автоматизации системных задач Windows. Скрипты PowerShell позволяют эффективно управлять системными компонентами: удалять предустановленные приложения, контролировать работу служб и ресурсоемких процессов. Особенно полезными оказались сценарии для работы с файловой системой, включая групповое переименование файлов, поиск по различным критериям и управление папками. Автоматизация этих задач существенно повышает производительность системного администрирования. Возможность создания сценариев для очистки устаревших файлов демонстрирует практический потенциал PowerShell в поддержании порядка на дисках и оптимизации работы системы.