МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ДГТУ)

Кафедра Кибербезопасность информационных систем

Разработал: Язвинская Н.Н.

ОПЕРАЦИОННЫЕ СИСТЕМЫ
Лабораторный практикум – Блок 1

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данном пособии представлен лабораторный практикум по дисциплине Операционные системы (ОС). Цели лабораторных занятий – это формирование у будущих специалистов 10.05.01 «Компьютерная безопасность» и бакалавров 09.03.01 «Информатика и вычислительная техника» систематического и целостного представления о значении и месте операционных систем в системном программном обеспечении вычислительных систем, об основных способах инсталляции, настроек и поддержки системных программных продуктов**.** Задачи лабораторных занятий: практическое освоение пользовательского интерфейса современных операционных систем; изучение взаимодействия аппаратных и программных средств на различных уровнях; изучение различных функциональных компонент современных операционных систем; изучение принципов управления различными ресурсами вычислительной системы и структурами данных.

Для полного освоения курса ОС необходимо последовательно выполнить все задания каждой работы, предварительно ознакомившись с теоретическим материалом курса лекций. Каждая лабораторная работа в данном пособии представляет собой решение отдельной проблемы для операционных систем семейства Windows.

В результате выполнения лабораторных работ по дисциплине ОС у студентов формируются следующие знания и навыки: основные функциональные компоненты ОС; средства мониторинга ОС; способы выбора ОС; способы реализации информационных систем и устройств в ОС; навыки отладки ОС и ее компонентов, эксплуатации современных ОС и решения поставленных задач в ОС; принципы работы основных подсистем ОС и способы защиты от несанкционированного доступа; пользовательский интерфейс современных ОС; навыки по разработке программного обеспечения на базе ОС; принципы анализа и оценки эффективности функционирования ОС и ее компонентов; навыки инсталляции и настройки параметров программного обеспечения ОС.

Лабораторные работы, представленные в пособии можно выполнять не только на базе лабораторий университета, но и дома при наличии соответствующей операционной системы на персональном компьютере. По результатам каждой лабораторной работы должен быть сформирован отчет, содержащий все команды и файлы, а также снимок экрана их выполнения. Каждая лабораторная работа содержит краткие теоретические сведения, которые являются дополнительным материалом к курсу лекций. В конце каждой работы есть вопросы для самоконтроля студента.

ТРЕБОВАНИЯ ПО ОФОРМЛЕНИЮ ОТЧЕТОВ ЛАБОРАТОРНЫХ РАБОТ

Отчет по лабораторной работе должен быть выполнен в редакторе MS Word и оформлен согласно требованиям.

Требования по форматированию:

- шрифт TimesNewRoman
- интервал – полуторный
- поля левое – 3 см., правое – 1,5 см., верхнее и нижнее – 2 см
- абзацный отступ – 1,25
- текст должен быть выравнен по ширине.

Отчет должен содержать: номер, тему лабораторной работы, кто выполнил (ФИО, группа) и проверил (ФИО, подпись, дата), цель работы и далее описанный процесс выполнения работы (ХОД РАБОТЫ), образец в Приложении А.

В отчет необходимо вставлять скриншоты выполненной работы и добавлять описание к ним.

Каждый рисунок должен располагаться по центру страницы, иметь подпись (Рисунок 1 – Окно Windows) и ссылку на него в тексте.

Если в Методике выполнения приводится таблица с заданиями по номерам, то номер задания соответствует списочному номеру студента в группе.

В конце отчета приводятся выводы о проделанной работе (если требуется) и ответ на контрольный вопрос письменно, номер вопроса соответствует номеру студента в списке группы. Ответы на остальные контрольные вопросы готовятся устно.

Отчет защищается выполнением практических заданий согласно Методике выполнения и устным ответом на контрольные вопросы по выбору преподавателя.

Лабораторная работа оценивается в промежутке от 0 до 100 баллов.

ЛАБОРАТОРНАЯ РАБОТА №1 (2 часа)
ИНТЕРПРЕТАТОР КОМАНДНОЙ СТРОКИ ОС MS WINDOWS

Цель работы – знакомство с возможностями интерпретатора командной строки и командами MS Windows

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Оболочка (интерпретатор) командной строки command.com/cmd.exe

Во всех версиях ОС Windows поддерживается интерактивная оболочка командной строки (command shell) и определенный набор утилит командной строки (количество и состав этих утилит зависит от версии ОС).

Механизм работы оболочек командной строки в разных системах одинаков: в ответ на приглашение ("подсказку", prompt), выдаваемое находящейся в ожидании оболочкой, пользователь вводит некоторую команду (функциональность этой команды может быть реализована либо самой оболочкой, либо определенной внешней утилитой), оболочка выполняет ее, при необходимости выводя на экран какую-либо информацию, после чего снова выводит приглашение и ожидает ввода следующей команды.

Оболочка представляет собой построчный интерпретатор простого языка сентенциального (директивного) программирования, в качестве операторов которого могут использоваться исполняемые программы.

Наряду с интерактивным режимом работы оболочки, как правило, поддерживают и пакетный режим, в котором система последовательно выполняет команды, записанные в текстовом файле-сценарии. Оболочка Windows не является исключением, с точки зрения программирования язык командных файлов Windows может быть охарактеризован следующим образом:

-реализация сентенциальной (директивной) парадигмы программирования;

-выполнение в режиме построчной интерпретации;

-наличие управляющих конструкций;

-поддержка нескольких видов циклов (в том числе специальных циклов для обработки текстовых файлов);

-наличие оператора присваивания (установки значения переменной);

-возможность использования внешних программ (команд) операционной системы в качестве операторов и обработки их кодов возврата;

-наличие нетипизированных переменных, которые декларируются первым упоминанием (значения переменных могут интерпретироваться как числа и использоваться в выражениях целочисленной арифметики).

Оболочка командной строки представляется интерпретатором Cmd.exe.

Итак, учитывая сказанное выше, можно сделать вывод: оболочка командной строки cmd.exe и командные файлы – наиболее универсальные и простые в изучении

средства автоматизации работы в Windows, доступные во всех версиях операционной системы.

1.2 Поддержка языков сценариев. Сервер сценариев Windows Script

Следующим шагом в развитии средств и технологий автоматизации в ОС Windows стало появление сервера сценариев Windows Script Host (WSH). Этот инструмент разработан для всех версий Windows и позволяет непосредственно в ОС выполнять сценарии на полноценных языках сценариев (по умолчанию, VBScript и JScript), которые до этого были доступны только внутри HTML-страниц и работали в контексте безопасности веб-браузера (в силу этого подобные сценарии, например, могли не иметь доступа к файловой системе локального компьютера).

По сравнению с командными файлами интерпретатора cmd.exe сценарии WSH имеют несколько преимуществ.

Во-первых, VBScript и JScript – это полноценные алгоритмические языки, имеющие встроенные функции и методы для обработки символьных строк, выполнения математических операций, обработки исключительных ситуаций и т.д.; кроме того, для написания сценариев WSH может использоваться любой другой язык сценариев (например, широко распространенный в Unix-системах Perl), для которого установлен соответствующий модуль поддержки.

Во-вторых, WSH поддерживает несколько собственных объектов, свойства и методы которых позволяют решать некоторые часто возникающие повседневные задачи администратора операционной системы: работа с сетевыми ресурсами, переменными среды, системным реестром, ярлыками и специальными папками Windows, запуск и управление работой других приложений.

В-третьих, из сценариев WSH можно обращаться к службам любых приложений-серверов автоматизации (например, программ из пакета MS Office), которые регистрируют в ОС свои объекты.

Наконец, сценарии WSH позволяют работать с объектами информационной модели Windows Management Instrumentation (WMI), обеспечивающей программный интерфейс управления всеми компонентами операционной модели, а также с объектами службы каталогов Active Directory Service Interface.

Следует также отметить, что технология WSH поддерживается в Windows уже довольно давно, в Интернете (в том числе на сайте Microsoft) можно найти множество готовых сценариев.

1.3 Командная оболочка Microsoft PowerShell

С одной стороны функциональности и гибкости языка оболочки cmd.exe явно недостаточно, а с другой стороны сценарии WSH, работающие с объектными моделями ADSI и WMI, слишком сложны для пользователей среднего уровня и начинающих администраторов.

Перед разработчиками новой оболочки, получившей название Windows PowerShell, стояли следующие основные цели:

-применение командной строки в качестве основного интерфейса администрирования;

-реализация модели ObjectFlow (элементом обмена информации является объект);

-переработка существующих команд, утилит и оболочки;

-интеграция командной строки, объектов COM, WMI и .NET;

-работа с произвольными источниками данных в командной строке по принципу файловой системы.

Самая важная идея, заложенная в PowerShell, состоит в том, что в командной строке вывод результатов команды представляет собой не текст (в смысле последовательности символов), а объект (данные вместе со свойственными им методами). В силу этого работать в PowerShell становится проще, чем в традиционных оболочках, так как не нужно выполнять никаких манипуляций по выделению нужной информации из символьного потока.

Отметим, что PowerShell одновременно является и оболочкой командной строки (пользователь работает в интерактивном режиме) и средой выполнения сценариев, которые пишутся на специальном языке PowerShell.

В целом, оболочка PowerShell намного удобнее и мощнее своих предшественников (cmd.exe и WSH).

2 Оболочка командной строки Windows. Интерпретатор Cmd.exe

Рассматриваются внутренние команды, поддерживаемые интерпретатором Cmd.exe, и наиболее часто используемые внешние команды (утилиты командной строки). Описываются механизмы перенаправления ввода/вывода, конвейеризации и условного выполнения команд.

В ОС Windows, как и в других ОС, интерактивные (набираемые с клавиатуры и сразу же выполняемые) команды выполняются с помощью так называемого командного интерпретатора, иначе называемого командным процессором или оболочкой командной строки (command shell). Начиная с версии Windows NT, в операционной системе реализован интерпретатор команд Cmd.exe, обладающий гораздо более широкими возможностями.

2.1 Запуск оболочки

В Windows файл Cmd.exe, как и другие исполняемые файлы, соответствующие внешним командам ОС, находятся в каталоге %SystemRoot%\SYSTEM32 (значением переменной среды %SystemRoot% является системный каталог Windows, обычно C:\Windows или C:\WinNT). Для запуска командного интерпретатора (открытия нового сеанса командной строки) можно выбрать пункт Выполнить… (Run) в меню Пуск (Start), ввести имя файла Cmd.exe и нажать кнопку OK. В результате откроется новое окно, в котором можно запускать команды и видеть результат их работы.

2.2 Настройка параметров командного окна интерпретатора

У утилиты командной строки, которая поставляется в виде стандартного приложения ОС Windows, имеется свой набор опций и параметров настройки. Один из способов просмотра этих опций – использование пункта Свойства управляющего меню окна (нажать правой кнопкой мыши на заголовок окна). В окне свойств будут доступны четыре вкладки с опциями: общие, шрифт, расположение и цвета.

2.3 Внутренние и внешние команды. Структура команд

Некоторые команды распознаются и выполняются непосредственно самим командным интерпретатором — такие команды называются **внутренними** (например, `COPY` или `DIR`). Другие команды ОС представляют собой отдельные программы, расположенные по умолчанию в том же каталоге, что и Cmd.exe, которые Windows загружает и выполняет аналогично другим программам. Такие команды называются внешними (например, `MORE` или `XCOPY`).

Рассмотрим структуру самой командной строки и принцип работы с ней. Для того, чтобы выполнить команду, после приглашения командной строки (например, `C:\>`) следует ввести имя этой команды (регистр не важен), ее параметры и ключи (если они необходимы) и нажать клавишу <Enter>. Например:

```
C:\>COPY C:\myfile.txt A:\ /V
```

Имя команды здесь — `COPY`, параметры — `C:\myfile.txt` и `A:\`, а ключом является `/V`. Отметим, что в некоторых командах ключи могут начинаться не с символа /, а с символа – (минус), например, `-V`.

Многие команды Windows имеют большое количество дополнительных параметров и ключей, запомнить которые зачастую бывает трудно. Большинство команд снабжено встроенной справкой, в которой кратко описываются назначение и синтаксис данной команды. Получить доступ к такой справке можно путем ввода команды с ключом `/?`. Например, если выполнить команду `ATTRIB /?`, то в окне MS-DOS мы увидим следующий текст: Отображение и изменение атрибутов файлов.

ATTRIB [+R|-R] [+A|-A] [+S|-S] [+H|-H] [[диск:][путь]имя_файла] [/S]

\+ Установка атрибута.

\- Снятие атрибута.

R Атрибут "Только чтение".

A Атрибут "Архивный".

S Атрибут "Системный".

H Атрибут "Скрытый".

/S Обработка файлов во всех вложенных папках указанного пути.

Для некоторых команд текст встроенной справки может быть довольно большим и не умещаться на одном экране. В этом случае помощь можно выводить последовательно по одному экрану с помощью команды `MORE` и символа конвейеризации |, например:

```
XCOPY /? | MORE
```

В этом случае после заполнения очередного экрана вывод помощи будет прерываться до нажатия любой клавиши. Кроме того, используя символы перенаправления вывода > и >>, можно текст, выводимый на экран, направить в текстовый файл для дальнейшего просмотра. Например, для вывода текста справки к команде XCOPY в текстовый файл xcopy.txt, используется следующая команда:

XCOPY /? > XCOPY.TXT

Замечание. Вместо имени файла можно указывать обозначения устройств компьютера. В Windows поддерживаются следующие **имена устройств**: `PRN` (принтер), CON (терминал: при вводе это клавиатура, при выводе - монитор), `NUL` (пустое устройство, все операции ввода/вывода для него игнорируются).

2.4 Перенаправление ввода/вывода и конвейеризация (композиция) команд

С помощью переназначения устройств ввода/вывода одна программа может направить свой вывод на вход другой или перехватить вывод другой программы, используя его в качестве своих входных данных. Таким образом, имеется возможность передавать информацию от процесса к процессу при минимальных программных издержках. Практически это означает, что для программ, которые используют стандартные входные и выходные устройства, ОС позволяет:

-выводить сообщения программ не на экран (стандартный выходной поток), а в файл или на принтер (перенаправление вывода);

-читать входные данные не с клавиатуры (стандартный входной поток), а из заранее подготовленного файла (перенаправление ввода);

-передавать сообщения, выводимые одной программой, в качестве входных данных для другой программы (конвейеризация или композиция команд).

Из командной строки эти возможности реализуются следующим образом. Для того, чтобы перенаправить текстовые сообщения, выводимые какой-либо командой, в текстовый файл, нужно использовать конструкцию

команда > имя_файла

Если при этом заданный для вывода файл уже существовал, то он перезаписывается, если не существовал — создается. Можно также не создавать файл заново, а дописывать информацию, выводимую командой, в конец существующего файла. Для этого команда перенаправления вывода должна быть задана так:

команда >> имя_файла

С помощью символа < можно прочитать входные данные для заданной команды не с клавиатуры, а из определенного (заранее подготовленного) файла:

команда < имя_файла

Примеры команд перенаправления ввода/вывода:

Вывод встроенной справки для команды COPY в файл copy.txt:

COPY /? > copy.txt

Добавление текста справки для команды XCOPY в файл copy.txt:

XCOPY /? >> copy.txt

Вывод текущей даты в файл date.txt (DATE /T — это команда для просмотра и изменения системной даты, T ключ для получения только даты без запроса нового значения):

DATE /T > date.txt

Если при выполнении определенной команды возникает ошибка, то сообщение об этом по умолчанию выводится на экран. В случае необходимости сообщения об ошибках (стандартный поток ошибок) можно перенаправить в текстовый файл с помощью конструкции

команда 2> имя_файла

В этом случае стандартный вывод будет производиться на экран.

Также имеется возможность информационные сообщения и сообщения об ошибках выводить в один и тот же файл. Делается это следующим образом:

команда > имя_файла 2>&1

Например, в приведенной ниже команде стандартный выходной поток и стандартный поток ошибок перенаправляются в файл copy.txt:

XCOPY A:\1.txt C: > copy.txt 2>&1

Наконец, с помощью конструкции

команда1 | команда2 можно использовать сообщения, выводимые первой командой, в качестве входных данных для второй команды (конвейер команд).

Используя механизмы перенаправления ввода/вывода и конвейеризации, можно из командной строки посылать информацию на различные устройства и автоматизировать ответы на запросы, выдаваемые командами или программами, использующими стандартный ввод. Для решения таких задач служит команда

ECHO [сообщение], которая выводит сообщение на экран. Пример использования этой команды.

Удаление всех файлов в текущем каталоге без предупреждения (автоматический положительный ответ на запрос об удалении):

ECHO y | DEL *.*

2.5 Команды MORE и SORT

Одной из наиболее часто использующихся команд, для работы с которой применяется перенаправление ввода/вывода и конвейеризация, является `MORE`. Эта команда считывает стандартный ввод из конвейера или перенаправленного файла и выводит информацию частями, размер каждой из которых не больше размера экрана. Используется `MORE` обычно для просмотра длинных файлов. Возможны три варианта синтаксиса этой команды:

MORE [диск:][путь]имя_файла

MORE < [диск:][путь]имя_файла

имя_команды | MORE

Параметр `[диск:][путь]имя_файла` определяет расположение и имя файла с просматриваемыми на экране данными. Параметр `имя_команды` задает команду, вывод которой отображается на экране (например, `DIR` или команда `TYPE`,

использующаяся для вывода содержимого текстового файла на экран). Приведем два примера.

Для поэкранного просмотра текстового файла news.txt возможны следующие варианты команд:

MORE news.txt
MORE < news.txt
TYPE news.txt | MORE 15

Другой распространенной командой, использующей перенаправление ввода/вывода и конвейеризацию, является SORT. Эта команда работает как фильтр: она считывает символы в заданном столбце, упорядочивает их в возрастающем или убывающем порядке и выводит отсортированную информацию в файл, на экран или другое устройство. Возможны два варианта синтаксиса этой команды:

SORT [/R] [/+n] [[диск1:][путь1]файл1] [> [диск2:][путь2]файл2]
или
[команда |] SORT [/R] [/+n] [> [диск2:][путь2]файл2]

В первом случае параметр [диск1:][путь1]файл1 определяет имя файла, который нужно отсортировать. Во втором случае будут отсортированы выходные данные указанной команды. Если параметры файл1 или команда не заданы, то SORT будет считывать данные с устройства стандартного ввода.

Параметр [диск2:][путь2]файл2 задает файл, в который будет направляется сортированный вывод; если этот параметр не задан, то вывод будет направлен на устройство стандартного вывода.

По умолчанию сортировка выполняется в порядке возрастания. Ключ /R позволяет изменить порядок сортировки на обратный (от Z к A и затем от 9 до 0). Например, для поэкранного просмотра отсортированного в обратном порядке файла price.txt, нужно задать следующую команду:

SORT /R < price.txt |MORE

Ключ /+n задает сортировку в файле по символам n-го столбца. Например, /+10 означает, что сортировка должна осуществляться, начиная с 10-й позиции в каждой строке. По умолчанию файл сортируется по первому столбцу.

2.6 Условное выполнение и группировка команд

В командной строке Windows можно использовать специальные символы, которые позволяют вводить несколько команд одновременно и управлять работой команд в зависимости от результатов их выполнения. С помощью таких символов условной обработки можно содержание небольшого пакетного файла записать в одной строке и выполнить полученную составную команду.

Используя символ амперсанда &, можно разделить несколько утилит в одной командной строке, при этом они будут выполняться друг за другом. Например, если набрать команду DIR & PAUSE & COPY /? и нажать клавишу <Enter>, то вначале на экран будет выведено содержимое текущего каталога, а после нажатия любой клавиши — встроенная справка команды COPY.

Условная обработка команд в Windows осуществляется с помощью символов `&&` и `||` следующим образом. Двойной амперсанд `&&` запускает команду, стоящую за ним в командной строке, только в том случае, если команда, стоящая перед амперсандами была выполнена успешно. Например, если в корневом каталоге диска C: есть файл plan.txt, то выполнение строки `TYPE C:\plan.txt && DIR` приведет к выводу на экран этого файла и содержимого текущего каталога. Если же файл C:\plan.txt не существует, то команда `DIR` выполняться не будет.

Два символа `||` осуществляют в командной строке обратное действие, т.е. запускают команду, стоящую за этими символами, только в том случае, если команда, идущая перед ними, не была успешно выполнена. Таким образом, если в предыдущем примере файл C:\plan.txt будет отсутствовать, то в результате выполнения строки `TYPE C:\plan.txt || DIR` на экран выведется содержимое текущего каталога.

Отметим, что условная обработка действует только на ближайшую команду, то есть в строке

TYPE C:\plan.txt && DIR & COPY /?

команда `COPY /?` запустится в любом случае, независимо от результата выполнения команды `TYPE C:\plan.txt`.

Несколько утилит можно сгруппировать в командной строке с помощью *круглых скобок*.

Пример:

TYPE C:\plan.txt && DIR & COPY /?

TYPE C:\plan.txt && (DIR & COPY /?)

В первой из них символ условной обработки `&&` действует только на команду `DIR`, во второй — одновременно на две команды: `DIR` и `COPY`.

3 Команды для работы с файловой системой

Рассмотрим некоторые наиболее часто используемые команды для работы с файловой системой. Отметим сначала несколько особенностей определения путей к файлам в Windows.

Файловая система логически имеет древовидную структуру и имена файлов задаются в формате `[диск:][путь\]имя_файла`, то есть обязательным параметром является только имя файла. При этом, если путь начинается с символа `"\"`, то маршрут вычисляется от корневого каталога, иначе — от текущего каталога. Например, имя C:123.txt задает файл 123.txt в текущем каталоге на диске C:, имя C:\123.txt — файл 123.txt в корневом каталоге на диске C:, имя ABC\123.txt — файл 123.txt в подкаталоге ABC текущего каталога.

Существуют особые обозначения для текущего каталога и родительского каталогов. Текущий каталог обозначается символом . (точка), его родительский каталог — символами .. (две точки). Например, если текущим каталогом является C:\WINDOWS, то путь к файлу autoexec.bat в корневом каталоге диска C: может быть записан в виде ..\autoexec.bat.

В именах файлов (но не дисков или каталогов) можно применять так называемые **групповые символы** или шаблоны: ? (вопросительный знак) и * (звездочка). Символ * в имени файла означает произвольное количество любых допустимых символов, символ ? — один произвольный символ или его отсутствие. Скажем, под шаблон text??1.txt подходят, например, имена text121.txt и text11.txt, под шаблон text*.txt — имена text.txt, textab12.txt, а под шаблон text.* — все файлы с именем text и произвольным расширением.

Для того, чтобы использовать длинные имена файлов при работе с командной строкой, их нужно заключать в двойные кавычки. Например, чтобы запустить файл с именем 'Мое приложение.exe' из каталога 'Мои документы', нужно в командной строке набрать `"C:\Мои документы\Мое приложение.exe"` и нажать клавишу <Enter>.

Текущий каталог можно изменить с помощью команды CD [диск:][путь\]. Путь к требуемому каталогу указывается с учетом приведенных выше замечаний. Например, команда `CD \` выполняет переход в корневой каталог текущего диска. Если запустить CD без параметров, то на экран будут выведены имена текущего диска и каталога.

Одной из наиболее часто повторяющихся задач при работе на компьютере является копирование и перемещение файлов из одного места в другое. Для копирования одного или нескольких файлов используется команда COPY.

Синтаксис этой команды:

COPY [/A|/B] источник [/A|/B] [+ источник [/A|/B] [+ ...]]
[результат [/A|/B]] [/V][/Y|/–Y]

Краткое описание параметров и ключей команды `COPY` приведено в (табл. 1).

Примеры использования команды `COPY`.

Копирование файла abc.txt из текущего каталога в каталог D:\PROGRAM под тем же именем:

COPY abc.txt D:\PROGRAM

Копирование файла abc.txt из текущего каталога в каталог D:\PROGRAM под новым именем def.txt:

COPY abc.txt D:\PROGRAM\def.txt

Копирование всех файлов с расширением txt с диска A: в каталог 'Мои документы' на диске C:

COPY A:*.txt "C:\Мои документы"

Если не задать в команде целевой файл, то команда COPY создаст копию файла-источника с тем же именем, датой и временем создания, что и исходный файл, и поместит новую копию в текущий каталог на текущем диске. Например, для того, чтобы скопировать все файлы из корневого каталога диска A: в текущий каталог, достаточно выполнить такую команду:

COPY A:*.*

Таблица 1.

Параметры и ключи команды COPY

Параметр	Описание
источник	Имя копируемого файла или файлов
/A	Файл является текстовым файлом ASCII, то есть конец файла обозначается символом с кодом ASCII 26 (<Ctrl>+<Z>)
/B	Файл является двоичным. Этот ключ указывает на то, что интерпретатор команд должен при копировании считывать из источника число байт, заданное размером в каталоге копируемого файла
результат	Каталог для размещения результата копирования и/или имя создаваемого файла
/V	Проверка правильности копирования путем сравнения файлов после копирования
/Y	Отключение режима запроса подтверждения на замену файлов
/-Y	Включение режима запроса подтверждения на замену файлов

Пример 1. Создания нового текстового файла и записи в него информации без использования текстового редактора.

Для решения задачи необходимо ввести команду `COPY CON my.txt`, которая будет копировать то, что набирается на клавиатуре в файл my.txt (если этот файл существовал, то он перезапишется, иначе — создастся). Для завершения ввода необходимо ввести символ конца файла, то есть нажать клавиши <Ctrl>+<Z>.

Команда `COPY` может также объединять (склеивать) несколько файлов в один. Для этого необходимо указать единственный результирующий файл и несколько исходных. Это достигается путем использования групповых знаков (? и *) или формата файл1 + файл2 + файл3. Например, для объединения файлов 1.txt и 2.txt в файл 3.txt можно задать следующую команду:

COPY 1.txt+2.txt 3.txt

Объединение всех файлов с расширением dat из текущего каталога в один файл all.dat может быть произведено так:

COPY /B *.dat all.dat

Ключ `/B` здесь используется для предотвращения усечения соединяемых файлов, так как при комбинировании файлов команда `COPY` по умолчанию считает файлами текстовыми.

Если имя целевого файла совпадает с именем одного из копируемых файлов (кроме первого), то исходное содержимое целевого файла теряется. Если имя целевого файла опущено, то в его качестве используется первый файл из списка. Например, команда `COPY 1.txt+2.txt` добавит к содержимому файла 1.txt содержимое файла 2.txt. Командой `COPY` можно воспользоваться и для присвоения какому-либо файлу

текущей даты и времени без модификации его содержимого. Для этого нужно ввести команду

COPY /B 1.txt +,,

Здесь запятые указывают на пропуск параметра приемника, что и приводит к требуемому результату.

Команда `COPY` имеет и свои недостатки. Например, с ее помощью нельзя копировать скрытые и системные файлы, файлы нулевой длины, файлы из подкаталогов. Кроме того, если при копировании группы файлов COPY встретит файл, который в данный момент нельзя скопировать (например, он занят другим приложением), то процесс копирования полностью прервется, и остальные файлы не будут скопированы.

Указанные при описании команды `COPY` проблемы можно решить с помощью команды `XCOPY`, которая предоставляет намного больше возможностей при копировании. Необходимо отметить, правда, что `XCOPY` может работать только с файлами и каталогами, но не с **устройствами**.

Синтаксис команды: XCOPY источник [результат] [ключи]

Команда `XCOPY` имеет множество ключей, далее приведены лишь некоторых из них. Ключ `/D[:[дата]]` позволяет копировать только файлы, измененные не ранее указанной даты. Если параметр дата не указан, то копирование будет производиться только если источник новее результата. Например, команда XCOPY "C:\Мои документы*.*" "D:\BACKUP\Мои документы" /D скопирует в каталог 'D:\BACKUP\Мои документы' только те файлы из каталога 'C:\Мои документы', которые были изменены со времени последнего подобного копирования или которых вообще не было в 'D:\BACKUP\Мои документы'.

Ключ `/S` позволяет копировать все непустые подкаталоги в каталоге-источнике. С помощью же ключа `/E` можно копировать вообще все подкаталоги, включая и пустые.

Если указан ключ `/C`, то копирование будет продолжаться даже в случае возникновения ошибок. Это бывает очень полезным при операциях копирования, производимых над группами файлов, например, при резервном копировании данных.

Ключ `/I` важен для случая, когда копируются несколько файлов, а файл назначения отсутствует. При задании этого ключа команда `XCOPY` считает, что файл назначения должен быть каталогом. Например, если задать ключ `/I` в команде копирования всех файлов с расширением txt из текущего каталога в несуществующий еще подкаталог TEXT, XCOPY *.txt TEXT /I то подкаталог TEXT будет создан без дополнительных запросов.

Ключи `/Q`, `/F` и `/L` отвечают за режим отображения при копировании. При задании ключа `/Q` имена файлов при копировании не отображаются, ключа `/F` — отображаются полные пути источника и результата. Ключ `/L` обозначает, что отображаются только файлы, которые должны быть скопированы (при этом само копирование не производится).

С помощью ключа `/H` можно копировать скрытые и системные файлы, а с

помощью ключа /R — заменять файлы с атрибутом "Только для чтения". Например, для копирования всех файлов из корневого каталога диска C: (включая системные и скрытые) в каталог SYS на диске D:, нужно ввести следующую команду:

XCOPY C:*.* D:\SYS /H

Ключ /T позволяет применять XCOPY для копирования только структуры каталогов источника, без дублирования находящихся в этих каталогах файлов, причем пустые каталоги и подкаталоги не включаются. Для того, чтобы все же включить пустые каталоги и подкаталоги, нужно использовать комбинацию ключей /T /E.

Используя XCOPY можно при копировании обновлять только уже существующие файлы (новые файлы при этом не записываются). Для этого применяется ключ /U. Например, если в каталоге C:\2 находились файлы a.txt и b.txt, а в каталоге C:\1 — файлы a.txt, b.txt, c.txt и d.txt, то после выполнения команды:

XCOPY C:\1 C:\2 /U

в каталоге C:\2 по-прежнему останутся лишь два файла a.txt и b.txt, содержимое которых будет заменено содержимым соответствующих файлов из каталога C:\1. Если с помощью XCOPY копировался файл с атрибутом "Только для чтения", то по умолчанию у файла-копии этот атрибут снимется. Для того, чтобы копировать не только данные, но и полностью атрибуты файла, необходимо использовать ключ /K.

Ключи /Y и /-Y определяют, нужно ли запрашивать подтверждение перед заменой файлов при копировании. /Y означает, что такой запрос нужен, /-Y — не нужен.

Команда: DIR [диск:][путь][имя_файла] [ключи]используется для вывода информации о содержимом дисков и каталогов. Параметр[диск:][путь] задает диск и каталог, содержимое которого нужно вывести на экран. Параметр [имя_файла] задает файл или группу файлов, которые нужно включить в список.

Например, команда DIR C:*.bat выведет на экран все файлы с расширением bat в корневом каталоге диска C:. Если задать эту команду без параметров, то выводится метка диска и его серийный номер, имена (в коротком и длинном вариантах) файлов и подкаталогов, находящихся в текущем каталоге, а также дата и время их последней модификации. После этого выводится число файлов в каталоге, общий объем (в байтах), занимаемый файлами, и объем свободного пространства на диске. Например:

Том в устройстве C имеет метку PHYS1_PART2

Серийный номер тома: 366D-6107

Содержимое папки C:\aditor

. <ПАПКА> 25.01.15 17:15 .

.. <ПАПКА> 25.01.15 17:15 ..

HILITE DAT 1 082 18.09.16 18:55 hilite.dat

TEMPLT01 DAT 48 07.08.16 1:00 templt01.dat

TTABLE DAT 357 07.08.16 1:00 ttable.dat

ADITOR EXE 461 312 01.12.15 23:13 aditor.exe

README TXT 3 974 25.01.15 17:26 readme.txt

ADITOR HLP 24 594 08.10.16 23:12 aditor.hlp

ТЕКСТО~1 TXT 0 11.03.15 9:02 Текстовый файл.txt
11 файлов 533 647 байт
2 папок 143 261 696 байт свободно

С помощью ключей команды DIR можно задать различные режимы расположения, фильтрации и сортировки. Например, при использовании ключа /W перечень файлов выводится в широком формате с максимально возможным числом имен файлов или каталогов на каждой строке. Например: Том в устройстве C имеет метку PHYS1_PART2
Серийный номер тома: 366D-6107
Содержимое папки C:\aditor
[.] [..] TEMPLT02.DAT UNINST1.000 HILITE.DAT
TEMPLT01.DAT UNINST0.000 TTABLE.DAT ADITOR.EXE README.TXT
ADITOR.HLP ТЕКСТО~1.TXT

11 файлов 533 647 байт
2 папок 143 257 600 байт свободно

С помощью ключа /A[[:]атрибуты] можно вывести имена только тех каталогов и файлов, которые имеют заданные атрибуты (R — "Только чтение", A — "Архивный", S — "Системный", H — "Скрытый", префикс "–" имеет значение НЕ). Если ключ /A используется более чем с одним значением атрибута, будут выведены имена только тех файлов, у которых все атрибуты совпадают с заданными. Например, для вывода имен всех файлов в корневом каталоге диска C:, которые одновременно являются скрытыми и системными, можно задать команду

DIR C:\ /A:HS

а для вывода всех файлов, кроме скрытых — команду

DIR C:\ /A:-H

Отметим здесь, что атрибуту каталога соответствует буква D, и для того, чтобы, например, вывести список всех каталогов диска C:, нужно задать команду

DIR C: /A:D

Ключ /O[[:]сортировка] задает порядок сортировки содержимого каталога при выводе его командой DIR. Если этот ключ опущен, DIR печатает имена файлов и каталогов в том порядке, в котором они содержатся в каталоге. Если ключ /O задан, а параметр сортировка не указан, то DIR выводит имена в алфавитном порядке. В параметре сортировка можно использовать следующие значения: N — по имени (алфавитная), S — по размеру (начиная с меньших), E — по расширению (алфавитная), D — по дате (начиная с более старых), A — по дате загрузки (начиная с более старых), G — начать список с каталогов. Префикс "–" означает обратный порядок. Если задается более одного значения порядка сортировки, файлы сортируются по первому критерию, затем по второму и т.д.

Ключ /S означает вывод списка файлов из заданного каталога и его подкаталогов. Ключ /B перечисляет только названия каталогов и имена файлов (в длинном формате) по одному на строку, включая расширение. При этом выводится

только основная информация, без итоговой. Например:

templt02.dat
UNINST1.000
hilite.dat
templt01.dat
UNINST0.000
ttable.dat
aditor.exe
readme.txt
aditor.hlp
Текстовый файл.txt

Для создания нового каталога и удаления уже существующего пустого каталога используются команды MKDIR [диск:]путь и RMDIR [диск:]путь [ключи] соответственно (или их короткие аналоги MD и RD). Например:

MKDIR "C:\Примеры"
RMDIR "C:\Примеры"

Команда MKDIR не может быть выполнена, если каталог или файл с заданным именем уже существует. Команда RMDIR не будет выполнена, если удаляемый каталог не пустой.

Удалить один или несколько файлов можно с помощью команды

DEL [диск:][путь]имя_файла [ключи]

Для удаления сразу нескольких файлов используются групповые знаки ? и *. Ключ /S позволяет удалить указанные файлы из всех подкаталогов, ключ /F – принудительно удалить файлы, доступные только для чтения, ключ /A[[:]атрибуты] – отбирать файлы для удаления по атрибутам (аналогично ключу /A[[:]атрибуты] в команде DIR).

Переименовать файлы и каталоги можно с помощью команды RENAME (REN).

Синтаксис этой команды имеет следующий вид:

REN [диск:][путь][каталог1|файл1] [каталог2|файл2]

Здесь параметр каталог1|файл1 определяет название каталога/файла, которое нужно изменить, а каталог2|файл2 задает новое название каталога/файла. В любом параметре команды REN можно использовать групповые символы ? и *. При этом представленные шаблонами символы в параметре файл2 будут идентичны соответствующим символам в параметре файл1. Например, чтобы изменить у всех файлов с расширением txt в текущей директории расширение на doc, нужно ввести такую команду:

REN *.txt *.doc

Если файл с именем файл2 уже существует, то команда REN прекратит выполнение, и произойдет вывод сообщения, что файл уже существует или занят. Кроме того, в команде REN нельзя указать другой диск или каталог для создания результирующих каталога и файла. Для этой цели нужно использовать команду MOVE, предназначенную для переименования и перемещения файлов и каталогов.

Синтаксис команды для перемещения одного или более файлов имеет вид:

MOVE [/Y|/–Y] [диск:][путь]имя_файла1[,...] результирующий_файл

Синтаксис команды для переименования папки имеет вид:

MOVE [/Y|/–Y] [диск:][путь]каталог1 каталог2

Здесь параметр `результирующий_файл` задает новое размещение файла и может включать имя диска, двоеточие, имя каталога, либо их сочетание. Если перемещается только один файл, допускается указать его новое имя. Это позволяет сразу переместить и переименовать файл. Например, MOVE "C:\Мои документы\список.txt" D:\list.txt.

Если указан ключ /-Y, то при создании каталогов и замене файлов будет выдаваться запрос на подтверждение. Ключ /Y отменяет выдачу такого запроса.

4 Язык интерпретатора Cmd.exe. Командные файлы

Язык оболочки командной строки (shell language) в Windows реализован в виде командных (или пакетных) файлов. **Командный файл** в Windows — это обычный текстовый файл с расширением bat или cmd, в котором записаны допустимые команды ОС (как внешние, так и внутренние), а также некоторые дополнительные инструкции и ключевые слова, придающие командным файлам некоторое сходство с программами, написанными на языке программирования. Например, если записать в файл deltmp.bat следующие команды:

C:\

CD %TEMP%

DEL /F *.tmp и запустить его на выполнение (аналогично исполняемым файлам с расширением com или exe), то мы удалим все файлы во временной директории Windows. Таким образом, исполнение командного файла приводит к тому же результату, что и последовательный ввод записанных в нем команд. При этом не проводится никакой предварительной компиляции или проверки синтаксиса кода; если встречается строка с ошибочной командой, то она игнорируется. Очевидно, что если приходится часто выполнять одни и те же действия, то использование командных файлов может сэкономить много времени.

4.1 Вывод сообщений и дублирование команд

По умолчанию команды пакетного файла перед исполнением выводятся на экран, что выглядит не очень эстетично. С помощью команды ECHO OFF можно отключить дублирование команд, идущих после нее (сама команда ECHO OFF при этом все же дублируется). Например,

REM Следующие две команды будут дублироваться на экране …

:: эта строка – такой же комментарий, как и
предыдущая DIR C:\

```
ECHO OFF
:: А остальные уже не будут
DIR D:\
```

Для восстановления режима дублирования используется команда ECHO ON. Кроме этого, можно отключить дублирование любой отдельной строки в командном файле, написав в начале этой строки символ @, например:

```
ECHO ON
:: Команда DIR C:\ дублируется на экране DIR C:\
:: А команда DIR D:\ — нет @DIR D:\
```

Таким образом, если поставить в самое начало файла команду @ECHO OFF,то это решит все проблемы с дублированием команд.

В пакетном файле можно выводить на экран строки с сообщениями.

Делается это с помощью команды ECHO сообщение Например:

```
@ECHO OFF
ECHO Привет!
```

Команда ECHO. (точка должна следовать непосредственно за словом "ECHO") выводит на экран **пустую строку**. Например:

```
@ECHO OFF
ECHO Привет!
ECHO.
ECHO Пока!
```

Часто бывает удобно для просмотра сообщений, выводимых из пакетного файла, предварительно полностью очистить экран командой CLS.

Используя механизм *перенаправления ввода/вывода* (символы > и >>), можно направить сообщения, выводимые командой ECHO, в определенный текстовый файл. Например:

```
@ECHO OFF
ECHO Привет! > hi.txt ECHO Пока! >> hi.txt
```

С помощью такого метода можно, скажем, заполнять файлы-протоколы с отчетом о произведенных действиях. Например:

```
@ECHO OFF
REM Попытка копирования
XCOPY C:\PROGRAMS D:\PROGRAMS /s
:: Добавление сообщения в файл report.txt в случае удачного завершения копирования
IF NOT ERRORLEVEL 1 ECHO Успешное копирование >> report.txt
```

Использование параметров командной строки

При запуске пакетных файлов в командной строке можно указывать произвольное число параметров, значения которых можно использовать внутри файла. Это позволяет, например, применять один и тот же командный файл для выполнения

команд с различными параметрами.

Для доступа из командного файла к параметрам командной строки применяются символы %0, %1, …, %9 или %*. При этом вместо %0 подставляется имя выполняемого пакетного файла, вместо %1, %2, …, %9

— значения первых девяти параметров командной строки соответственно, а вместо %* — все аргументы. Если в командной строке при вызове пакетного файла задано меньше девяти параметров, то "лишние" переменные из %1 – %9 замещаются пустыми строками. Рассмотрим следующий пример. Пусть имеется командный файл copier.bat следующего содержания:

```
@ECHO OFF CLS
ECHO Файл %0 копирует каталог %1 в %2
XCOPY %1 %2 /S
```

Если запустить его из командной строки с двумя параметрами, например copier.bat C:\Programs D:\Backup то на экран выведется сообщение

Файл copier.bat копирует каталог C:\Programs в D:\Backup

и произойдет копирование каталога C:\Programs со всеми его подкаталогами в D:\Backup.

При необходимости можно использовать более девяти параметров командной строки. Это достигается с помощью команды `SHIFT`, которая изменяет значения замещаемых параметров с `%0` по `%9`, копируя каждый параметр в предыдущий, то есть значение `%1` копируется в `%0`, значение

`%2` – в `%1` и т.д. Замещаемому параметру `%9` присваивается значение параметра, следующего в командной строке за старым значением `%9`. Если же такой параметр не задан, то новое значение `%9` — пустая строка.

Пример 1. Пусть командный файл my.bat вызван из командной строки следующим образом:

my.bat p1 p2 p3

Тогда %0=my.bat, %1=p1, %2=p2, %3=p3, параметры %4 – %9 являются пустыми строками. После выполнения команды SHIFT значения замещаемых параметров изменятся следующим образом: %0=p1, %1=p2,

%2=p3, параметры %3 – %9 – пустые строки.

При включении расширенной обработки команд SHIFT поддерживает ключ /n, задающий начало сдвига параметров с номера n, где n может быть числом от 0 до 9.

Например, в следующей команде SHIFT /2 параметр %2 заменяется на %3,

%3 на %4 и т.д., а параметры %0 и %1 остаются без изменений.

Команда, обратная SHIFT (обратный сдвиг), отсутствует. После выполнения SHIFT уже нельзя восстановить параметр (%0), который был первым перед сдвигом. Если в командной строке задано больше десяти параметров, то команду SHIFT можно использовать несколько раз.

В командных файлах имеются некоторые возможности синтаксического анализа заменяемых параметров. Для параметра с номером n (%n) допустимы синтаксические конструкции (операторы), представленные в (табл. 2).

Таблица 2 - Операторы для заменяемых параметров

Операторы	Описание
%~Fn	Переменная %n расширяется до полного имени файла
%~Dn	Из переменной %n выделяется только имя диска
%~Pn	Из переменной %n выделяется только путь к файлу
%~Nn	Из переменной %n выделяется только имя файла
%~Xn	Из переменной %n выделяется расширение имени файла
%~Tn	Возвращается дата и время создания (модификации) файла
%~Zn	Возвращается размер файла в байтах
%~$PATH:n	Проводится поиск по каталогам, заданным в переменной среды PATH, и переменная %n заменяется на полное имя первого найденного файла. Если переменная PATH не определена или в результате поиска не найден ни один файл, эта конструкция заменяется на пустую строку. Естественно, здесь переменную PATH можно заменить на любое другое допустимое значение

Данные синтаксические конструкции можно объединять друг с другом, например:

%~DPn — из переменной %n выделяется имя диска и путь,

%~NXn — из переменной %n выделяется имя файла и расширение.

Пример 2. Пусть мы находимся в каталоге C:\TEXT и запускаем пакетный файл с параметром Рассказ.doc (%1=Рассказ.doc). Размер файла 2150 байт, дата создания 12.12.2015, время -12:55. Тогда применение операторов, описанных в табл. 1, к параметру %1 даст следующие результаты:

%~F1=C:\TEXT\Рассказ.doc
%~D1=C:
%~P1=\TEXT\
%~N1=Рассказ
%~X1=.doc
%~DP1=C:\TEXT\
%~NX1=Рассказ.doc
%~T1=12.12.2009 12:55
%~Z1=2150

Работа с переменными среды

Внутри командных файлов можно использовать так называемые **переменными среды** (или переменными окружения), каждая из которых хранится в оперативной

памяти, имеет свое уникальное имя, а ее значением является **строка**. Стандартные переменные среды автоматически инициализируются в процессе загрузки операционной системы. Такими переменными являются:

WINDIR, которая определяет расположение каталога Windows, TEMP, которая определяет путь к каталогу для хранения временных файлов Windows

- PATH, в которой хранится системный путь (путь поиска), то есть список каталогов, в которых система должна искать выполняемые файлы или файлы совместного доступа (например, динамические библиотеки).

Кроме того, в командных файлах с помощью команды `SET` можно объявлять собственные переменные среды.

Получение значения переменной

Для получения значения определенной переменной среды нужно заключить имя этой переменной в символы `%`. Например:

```
@ECHO OFF
CLS
:: Создание переменной MyVar SET MyVar=Привет
:: Изменение переменной SET MyVar=%MyVar%
ECHO Значение переменной MyVar: %MyVar%
:: Удаление переменной MyVar SET MyVar=
ECHO Значение переменной WinDir: %WinDir%
```

При запуске такого командного файла на экран выведется строка Значение переменной MyVar: Привет!

Значение переменной WinDir: C:\WINDOWS

Преобразования переменных как строк

С переменными среды в командных файлах можно производить некоторые манипуляции. Во-первых, над ними можно производить операцию конкатенации (соединения). Для этого нужно в команде `SET` просто написать рядом значения соединяемых переменных. Например, SET A=Раз

```
SET B=Два
SET C=%A%%B%
```

После выполнения в файле этих команд значением переменной C будет являться строка `'РазДва'`. Не следует для конкатенации использовать знак `+`, так как он будет воспринят просто в качестве символа. Например, после запуска файл следующего содержания

```
SET A=Раз SET B=Два SET C=A+B
ECHO Переменная C=%C% SET D=%A%+%B%
ECHO Переменная D=%D%
```

на экран выведутся две строки:

Переменная C=A+B Переменная D=Раз+Два

Во-вторых, из переменной среды можно **выделять подстроки** с помощью

конструкции %имя_переменной:~n1,n2%, где число n1 определяет смещение (количество пропускаемых символов) от начала (если n1 положительно) или от конца (если n1 отрицательно) соответствующей переменной среды, а число n2 – количество выделяемых символов (если n2 положительно) или количество последних символов в переменной, которые не войдут в выделяемую подстроку (если n2 отрицательно). Если указан только один отрицательный параметр -n, то будут извлечены последние n символов. Например, если в переменной хранится строка "21.12.2015" (символьное представление текущей даты при определенных региональных настройках), то после выполнения следующих команд

SET dd1=%DATE:~0,2% SET dd2=%DATE:~0,-8% SET mm=%DATE:~-7,2% SET yyyy=%DATE:~-4%

новые переменные будут иметь такие значения: %dd1%=21, %dd2%=21, %mm%=12, %yyyy%=2015.

В-третьих, можно выполнять процедуру замены подстрок с помощью конструкции %имя_переменной:s1=s2% (в результате будет возвращена строка, в которой каждое вхождение подстроки s1 в соответствующей переменной среды заменено на s2). Например, после выполнения команд

SET a=123456 SET b=%a:23=99%

в переменной b будет храниться строка "199456". Если параметр s2 не указан, то подстрока s1 будет удалена из выводимой строки, т.е. после выполнения команды

SET a=123456 SET b=%a:23=%

в переменной b будет храниться строка "1456".

Операции с переменными как с числами

При включенной расширенной обработке команд (этот режим в Windows используется по умолчанию) имеется **возможность рассматривать значения переменных среды как числа** и производить с ними арифметические вычисления (используются ТОЛЬКО целые числа). Для этого используется команда SET с ключом /A. Ниже приведен пример пакетного файла add.bat, складывающего два числа, заданных в качестве параметров командной строки, и выводящего полученную сумму на экран: @ECHO OFF

:: В переменной M будет храниться сумма SET /A M=%1+%2

ECHO Сумма %1 и %2 равна %M%

:: Удалим переменную M SET M=

В команде SET с ключом /A могут использоваться операции – (вычитание), * (умножение), / (деление нацело), % (остаток от деления). При использовании знака % в качестве знака операции в **командных файлах** он должен быть записан ДВАЖДЫ.

Рекомендуется при инициализации числовых переменных использовать ключ /A

SET /A col=0

Ввод значения переменной с клавиатуры

Ввод значения переменной при выполнении командного файла выполняется командой SET с ключом /P. Например, для ввода значения переменной M следует использовать команду

SET /P M=[введите M]

Текст подсказки [введите M] будет выведен на экран.

Локальные изменения переменных

Все изменения, производимые с помощью команды `SET` над переменными среды в командном файле, сохраняются и после завершения работы этого файла, но действуют только внутри текущего командного окна. Также имеется возможность локализовать изменения переменных среды внутри пакетного файла, то есть автоматически восстанавливать значения всех переменных в том виде, в каком они были **до начала запуска** этого файла. Для этого используются две команды: `SETLOCAL` и `ENDLOCAL`. Команда `SETLOCAL` определяет начало области локальных установок переменных среды. Другими словами, изменения среды, внесенные после выполнения `SETLOCAL`, будут являться локальными относительно текущего пакетного файла. Каждая команда `SETLOCAL` должна иметь соответствующую команду `ENDLOCAL` для восстановления прежних значений переменных среды. Изменения среды, внесенные после выполнения команды `ENDLOCAL`, уже не являются локальными относительно текущего пакетного файла; их прежние значения не будут восстановлены по завершении выполнения этого файла.

Связывание времени выполнения для переменных

При работе с составными выражениями (группы команд, заключенных в круглые скобки) нужно учитывать, что переменные среды в командных файлах используются в режиме раннего связывания. С точки зрения логики выполнения командного файла это может привести к ошибкам. Например, рассмотрим командный файл 1.bat со следующим содержимым:

SET a=1 ECHO a=%a% SET a=2 ECHO a=%a%

и командный файл 2.bat:

SET a=1 ECHO a=%a% (SET a=2

ECHO a=%a%)

Казалось бы, результат выполнения этих двух файлов должен быть одинаковым: на экран выведутся две строки: `"a=1" и "a=2"`. На самом же деле таким образом сработает только файл 1.bat, а файл 2.bat два раза выведет строку `"a=1".`

Данную ошибку можно обойти, если для получения значения переменной вместо знаков процента `(%)` использовать восклицательный знак (!) и предварительно включить режим связывания времени выполнения командой `SETLOCAL ENABLEDELAYEDEXPANSION`. Таким образом, для корректной работы файл 2.bat должен иметь следующий вид: SETLOCAL ENABLEDELAYEDEXPANSION

SET a=1

ECHO a=%a% (SET a=2 ECHO a=!a!)

ВНИМАНИЕ! Приведенный материал необходим для правильной работы команды цикла FOR и будет использован в командных файлах!

Приостановка выполнения командных файлов

Для того, чтобы вручную **прервать выполнение** запущенного bat- файла, нужно нажать клавиши <Ctrl>+<C> или <Ctrl>+<Break>. Однако часто бывает необходимо программно приостановить выполнение командного файла в определенной строке с выдачей запроса на нажатие любой клавиши. Это делается с помощью команды PAUSE. Перед запуском этой команды полезно с помощью команды `ECHO` информировать пользователя о действиях, которые он должен произвести. Например:

ECHO Вставьте дискету в дисковод A: и нажмите любую клавишу PAUSE

Команду `PAUSE` обязательно нужно использовать при выполнении потенциально опасных действий (удаление файлов, форматирование дисков и т.п.). Например,

ECHO Сейчас будут удалены все файлы в C:\Мои документы! ECHO Для отмены нажмите Ctrl-C

PAUSE

DEL "C:\Мои документы*.*"

Вызов внешних командных файлов

Из одного командного файла можно вызвать другой, просто указав его имя. Например:

@ECHO OFF CLS

REM Вывод списка log-файлов DIR C:*.log

:: Передача выполнения файлу f.bat f.bat

COPY A:*.* C:\ PAUSE

Однако в этом случае после выполнения вызванного файла управление в вызывающий файл не передается, то есть в приведенном примере команда

COPY A:*.* C:\

(и все следующие за ней команды) никогда не будет выполнена.

Для того, чтобы вызвать внешний командный файл с последующим возвратом в первоначальный файл, нужно использовать специальнуюкоманду CALL файл Например:

@ECHO OFF CLS

:: Вывод списка log-файлов DIR C:*.log

:: Передача выполнения файлу f.bat CALL f.bat

COPY A:*.* C:\ PAUSE

В этом случае после завершения работы файла f.bat управление вернется в первоначальный файл на строку, следующую за командой `CALL` (в нашем примере это команда `COPY A:\*.* C:\`).

Операторы перехода GOTO и вызова CALL

Командный файл может содержать метки и команды GOTO перехода к этим меткам. Любая строка, начинающаяся с двоеточия :, воспринимается при обработке командного файла как метка. Имя метки задается набором символов, следующих за двоеточием до первого пробела или конца строки.

Пример 3. Пусть имеется командный файл следующего содержания: @ECHO OFF

COPY %1 %2
GOTO Label1
ECHO Эта строка никогда не выполнится
:Label1
:: Продолжение выполнения DIR %2
После того, как в этом файле мы доходим до команды GOTO Label1
его выполнение продолжается со строки
:: Продолжение выполнения

В команде перехода внутри файла GOTO можно задавать в качестве метки перехода строку :EOF, которая передает управление в конец текущего пакетного файла (это позволяет легко выйти из пакетного файла без определения каких-либо меток в самом его конце).

Для перехода к метке внутри текущего командного файла кроме команды GOTO можно использовать и рассмотренную выше команду CALL:

CALL :метка аргументы

При вызове такой команды создается новый контекст текущего пакетного файла с заданными аргументами, и управление передается на инструкцию, расположенную сразу после метки. Для выхода из такого пакетного файла необходимо два раза достичь его конца. Первый выход возвращает управление на инструкцию, расположенную сразу после строки CALL, а второй выход завершает выполнение пакетного файла. Например, если запустить с параметром "Копия-1" командный файл следующего содержания:

@ECHO OFF ECHO %1
CALL :2 Копия-2
:2
ECHO %1
то на экран выведутся три строки:
Копия-1 Копия-2 Копия-1

Таким образом, подобное использование команды CALL похоже на вызов подпрограмм в языках высокого уровня.

Оператор проверки условия IF

С помощью команды IF … ELSE (ключевое слово ELSE может отсутствовать) в пакетных файлах можно выполнять обработку условий нескольких типов. При этом если заданное после IF условие принимает истинное значение, система выполняет следующую за условием команду (или несколько команд, заключенных в круглые

скобки), в противном случае выполняется команда (или несколько команд в скобках), следующие за ключевым словом ELSE.

Проверка значения переменной

Первый тип условия используется обычно для проверки значения переменной. Для этого применяются два варианта синтаксиса команды IF: IF [NOT] строка1==строка2 команда1 [ELSE команда2]

(квадратные скобки указывают на необязательность заключенных в них параметров) или

IF [/I] [NOT] строка1 оператор_сравнения строка2 команда

Рассмотрим сначала первый вариант. Условие строка1==строка2 (здесь необходимо писать именно два знака равенства – как и в программах на C/C++) считается истинным при точном совпадении обеих строк. Параметр NOT указывает на то, что заданная команда выполняется лишь в том случае, когда сравниваемые строки не совпадают.

Для *группировки команд* могут использоваться круглые скобки. Иногда использование круглых скобок необходимо для правильной работы команды if…else – например для вывода на экран наибольшего из двух параметров, с которыми запущен командный файл, следует использовать оператор

if %1 GTR %2 (echo %1) else (echo %2)

Строки могут быть литеральными или представлять собой значения переменных (например, %1 или %TEMP%). Кавычки для литеральных строк **не требуются**. Например,

IF %1==%2 ECHO Параметры совпадают!

IF %1==windows ECHO значение первого параметра - windows

Отметим, что при сравнении строк, заданных переменными, следует проявлять определенную осторожность. Дело в том, что значение переменной может оказаться пустой строкой, и тогда может возникнуть ситуация, при которой выполнение командного файла аварийно завершится. Например, если вы не определили с помощью команды SET переменную MyVar, а в файле имеется условный оператор типа

IF %MyVar%==C:\ ECHO Ура!!!, то в процессе выполнения вместо

%MyVar% подставится пустая строка и возникнет синтаксическая ошибка. Такая же ситуация может возникнуть, если одна из сравниваемых строк является значением параметра командной строки, так как этот параметр может быть не указан при запуске командного файла. Поэтому при сравнении строк нужно приписывать к ним в начале какой-нибудь символ, например:

IF -%MyVar%==-C:\ ECHO Ура!!!

С помощью команд IF и SHIFT можно в цикле обрабатывать все параметры командной строки файла, даже не зная заранее их количества. Например, следующий командный файл (назовем его primer.bat) выводит на экран имя запускаемого файла и все параметры командной строки: @ECHO OFF

ECHO Выполняется файл: %0 ECHO.

ECHO Файл запущен со следующими параметрами...

:: Начало цикла
:BegLoop
IF -%1==- GOTO ExitLoop ECHO %1
:: Сдвиг параметров SHIFT
:: Переход на начало цикла GOTO BegLoop
:ExitLoop
:: Выход из цикла ECHO.
ECHO Все.

Если запустить primer.bat с четырьмя параметрами:
primer.bat A B C D то в результате выполнения на экран выведется следующая информация:
Выполняется файл: primer.bat
Файл запущен со следующими параметрами: A
B C D
Все.
Рассмотрим теперь оператор `IF` в следующем виде: IF [/I] строка1 оператор_сравнения строка2 команда
Синтаксис и значение `операторов_сравнения` представлены в (табл. 3).

Таблица 3 - Операторы сравнения в IF

Оператор	Значение
EQL	Равно
NEQ	Не равно
LSS	Меньше
LEQ	Меньше или равно
GTR	Больше
GEQ	Больше или равно

Пример 4. использования операторов сравнения:
@ECHO OFF CLS
IF -%1 EQL –Вася ECHO Привет, Вася!
IF -%1 NEQ –Вася ECHO Привет, но Вы не Вася!
Ключ `/I`, если он указан, задает сравнение текстовых строк **без учета регистра**. Ключ `/I` можно также использовать и в форме `строка1==строка2` команды `IF`. Например, условие
IF /I DOS==dos … будет истинным.

Проверка существования заданного файла

Второй способ использования команды IF — это проверка существования заданного файла. Синтаксис для этого случая имеет вид:

IF [NOT] EXIST файл команда1 [ELSE команда2]

Условие считается истинным, если указанный файл существует. Кавычки для имени файла не требуются. Приведем пример командного файла, в котором с помощью такого варианта команды IF проверяется наличие файла, указанного в качестве параметра командной строки. @ECHO OFF

IF -%1==- GOTO NoFileSpecified

IF NOT EXIST %1 GOTO FileNotExist

:: Вывод сообщения о найденном файле ECHO Файл '%1' найден.

GOTO :EOF

:NoFileSpecified

:: Файл запущен без параметров

ECHO В командной строке не указано имя файла. GOTO :EOF

:FileNotExist

:: Параметр командной строки задан, но файл не найден ECHO Файл '%1' не найден.

Проверка наличия переменной среды

Аналогично файлам команда IF позволяет проверить наличие в системе определенной переменной среды:

IF DEFINED переменная команда1 [ELSE команда2]

Здесь условие DEFINED применяется подобно условию EXISTS наличия заданного файла, но принимает в качестве аргумента имя переменной среды и возвращает истинное значение, если эта переменная определена. Например:

@ECHO OFF CLS

IF DEFINED MyVar GOTO :VarExists ECHO Переменная MyVar не определена GOTO :EOF

:VarExists

ECHO Переменная MyVar определена, ECHO ее значение равно %MyVar%

Проверка кода завершения предыдущей команды

Еще один способ использования команды IF — это проверка кода завершения (кода выхода) предыдущей команды. Синтаксис для IF в этом случае имеет следующий вид:

IF [NOT] ERRORLEVEL число команда1 [ELSE команда2]

Здесь условие считается истинным, если последняя запущенная команда или программа завершилась с кодом возврата, равным либо превышающим указанное

число.

Рассмотрим командный файл, который копирует файл my.txt на диск C: без вывода на экран сообщений о копировании, а в случае возникновения какой-либо ошибки выдает предупреждение:

@ECHO OFF

XCOPY my.txt C:\ > NUL

:: Проверка кода завершения копирования IF ERRORLEVEL 1 GOTO ErrOccurred ECHO Копирование выполнено без ошибок. GOTO :EOF

:ErrOccurred

ECHO При выполнении команды XCOPY возникла ошибка!

В операторе `IF ERRORLEVEL ...` можно также применять операторы сравнения чисел, приведенные в табл. 2. Например:

IF ERRORLEVEL LEQ 1 GOTO Case1

Замечание. Иногда более удобным для работы с кодами завершения программ может оказаться использование переменной `%ERRORLEVEL%`. (строковое представление текущего значения кода ошибки `ERRORLEVEL`).

Организация циклов

В командных файлах для организации циклов используются несколько разновидностей оператора `FOR`, которые обеспечивают следующие функции:

- выполнение заданной команды для всех элементов указанного множества;
- выполнение заданной команды для всех подходящих имен файлов; выполнение заданной команды для всех подходящих имен каталогов;
- выполнение заданной команды для определенного каталога, а также всех его подкаталогов;
- получение последовательности чисел с заданными началом, концом и шагом приращения;
- чтение и обработка строк из текстового файла; обработка строк вывода определенной команды.

Цикл FOR … IN … DO …

Самый простой вариант синтаксиса команды `FOR` для командных файлов имеет следующий вид:

FOR %%переменная IN (множество) DO команда [параметры]

Внимание!

Перед названием переменной должны стоять именно два знака процента `(%%)`, а не один, как это было при использовании команды `FOR` непосредственно из командной строки!

Пример 5. Если в командном файле заданы строки @ECHO OFF

FOR %%i IN (Раз, Два, Три) DO ECHO %%i

то в результате его выполнения на экран будет выведено следующее: Раз

Два Три

Параметр `множество` в команде `FOR` задает одну или более текстовых строк, разделенных запятыми, которые необходимо обработать с помощью заданной команды. Скобки здесь **обязательны**. Параметр `команда [параметры]` задает команду, выполняемую для каждого элемента множества, при этом вложенность команд `FOR` на одной строке **не допускается**. Если в строке, входящей во множество, используется запятая, то значение этой строки нужно заключить в кавычки. Например, в результате выполнения файла с командами

@ECHO OFF

FOR %%i IN ("Раз,Два",Три) DO ECHO %%i

на экран будет выведено Раз,Два

Три

Параметр `%%переменная` представляет подставляемую переменную (счетчик цикла), причем здесь могут использоваться только имена переменных, **состоящие из одной буквы**. При выполнении команда `FOR` заменяет подставляемую переменную текстом каждой строки в заданном множестве, пока команда, стоящая после ключевого слова `DO`, не обработает все такие строки.

Замечание. Чтобы избежать путаницы с параметрами командного файла %0 — %9, для переменных следует использовать любые символы кроме 0 – 9.

Параметр множество в команде FOR может также представлять одну или несколько групп файлов. Например, чтобы вывести в файл список всех файлов с расширениями txt и prn, находящихся в каталоге C:\TEXT, без использования команды DIR, можно использовать командный файл следующего содержания:

@ECHO OFF

FOR %%f IN (C:\TEXT*.txt C:\TEXT*.prn) DO ECHO %%f >> list.txt

При таком использовании команды FOR процесс обработки продолжается, пока не обработаются все файлы (или группы файлов), указанные во множестве.

Цикл FOR /D … IN … DO …

Следующий вариант команды `FOR` реализуется с помощью ключа /D (directory – каталог):

FOR /D %переменная IN (набор) DO команда [параметры]

В случае, если набор содержит подстановочные знаки, то команда выполняется для всех подходящих имен каталогов, а не имен файлов. Скажем, выполнив следующий командный файл:

@ECHO OFF CLS

FOR /D %%f IN (C:*.*) DO ECHO %%f

мы получим список всех каталогов на диске C:, например: C:\Arc

C:\CYR C:\MSCAN

C:\Program Files C:\TEMP C:\WINNT

Цикл FOR /R … IN … DO …

С помощью ключа /R можно задать **рекурсию** в команде FOR: FOR /R [[диск:]путь] %переменная IN (набор)

DO команда [параметры]

В этом случае заданная команда выполняется для каталога [диск:]путь, а также для **всех подкаталогов** этого пути. Если после ключа R не указано имя каталога, то выполнение команды начинается с текущего каталога.

Пример 6. Для распечатки всех файлов с расширением txt в текущем каталоге и всех его подкаталогах можно использовать следующий пакетный файл:

```
@ECHO OFF CLS
FOR /R %%f IN (*.txt) DO PRINT %%f
```

Если вместо набора указана только точка (.), то команда проверяет все подкаталоги текущего каталога. Например, если мы находимся в каталоге C:\TEXT с двумя подкаталогами BOOKS и ARTICLES, то в результате выполнения файла:

```
@ECHO OFF CLS
FOR /R %%f IN (.) DO ECHO %%f
```

на экран выведутся три строки:

C:\TEXT\. C:\TEXT\BOOKS\. C:\TEXT\ARTICLES\.

Цикл FOR /L … IN … DO …

Ключ /L позволяет реализовать с помощью команды FOR цикл со счетчиком, в этом случае синтаксис имеет следующий: вид:

FOR /L %переменная IN (начало,шаг,конец) DO команда [параметры] Здесь заданная после ключевого слова IN тройка (начало, шаг, конец) задает последовательность чисел с заданными началом, концом и шагом приращения. Например, тройка (1, 1, 5) порождает последовательность (1 2 3 4 5), а тройка (5, -1, 1) - последовательность (5 4 3 2 1). Например, в результате выполнения следующего командного файла: @ECHO OFF CLS

```
FOR /L %%f IN (1,1,5) DO ECHO %%f
```

переменная цикла %%f получит значения от 1 до 5, и на экран будут выведены пять чисел:

```
1
2
3
4
5
```

Числа, получаемые в результате выполнения цикла FOR /L, можно использовать в **арифметических вычислениях**. Рассмотрим командный файл my.bat следующего содержания:

```
@ECHO OFF CLS
FOR /L %%f IN (1,1,5) DO CALL :2 %%f GOTO :EOF
:2
```

SET /A M=10*%1 ECHO 10*%1=%M%

В третьей строке в цикле происходит вызов нового контекста файла my.bat с текущим значением переменной цикла `%%f` в качестве параметра командной строки, причем управление передается на метку `:2` (см. описание `CALL` в разделе "Изменения в командах перехода"). В шестой строке переменная цикла умножается на десять, и результат записывается в переменную `M`. Таким образом, в результате выполнения этого файла выведется следующая информация:

10*1=10
10*2=20
10*3=30
10*4=40
10*5=50

Цикл FOR /F … IN … DO …

Самые широкие возможности имеет команда `FOR` с ключом `/F`: FOR /F ["ключи"] %переменная IN (набор) DO команда [параметры]

Здесь параметр набор содержит имена одного или нескольких файлов, которые по очереди открываются, читаются и обрабатываются. Обработка состоит в чтении файла, разбиении его на отдельные строки текста и выделении из каждой строки заданного числа подстрок. Затем найденная подстрока используется в качестве значения переменной при выполнении основного тела цикла (заданной команды).

По умолчанию ключ `/F` выделяет из каждой строки файла первое слово, очищенное от окружающих его пробелов. Пустые строки в файле пропускаются. Необязательный параметр `"ключи"` служит для переопределения заданных по умолчанию правил обработки строк. Ключи представляют собой заключенную в кавычки строку, содержащую приведенные в (табл. 4) ключевые слова:

Таблица 4 - Ключи в команде FOR /F

Ключ	Описание
`EOL=C`	Определение символа комментариев в начале строки (допускается задание только одного символа)
`SKIP=N`	Число пропускаемых при обработке строк в начале файла
`DELIMS=XXX`	Определение набора разделителей для замены заданных по умолчанию пробела и знака табуляции
`TOKENS=X,Y,M-N`	Определение номеров подстрок, выделяемых из каждой строки файла и передаваемых для выполнения в тело цикла

При использовании ключа `TOKENS=X,Y,M-N` создаются дополнительные переменные. Формат `M-N` представляет собой диапазон подстрок с номерами от `M` до `N`. Если последний символ в строке `TOKENS=` является звездочкой, то создается

дополнительная переменная, значением которой будет весь текст, оставшийся в строке после обработки последней подстроки.

Разберем применение этой команды на примере пакетного файла parser.bat, который производит разбор файла myfile.txt:

```
@ECHO OFF
IF NOT EXIST myfile.txt GOTO :NoFile
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN (myfile.txt) DO @ECHO
%%i %%j %%k GOTO :EOF
:NoFile
ECHO Не найден файл myfile.txt!
```

Здесь во второй строке производится проверка наличия файла myfile.txt; в случае отсутствия этого файла выводится предупреждающее сообщение. Команда `FOR` в третьей строке обрабатывает файл myfile.txt следующим образом:

Пропускаются все строки, которые начинаются с символа точки с запятой (`EOL=;`).

Вторая и третья подстроки из каждой строки передаются в тело

цикла, причем подстроки разделяются пробелами (по умолчанию) и/или запятыми (`DELIMS=,`).

В теле цикла переменная `%%i` используется для второй подстроки,

`%%j` — для третьей, а `%%k` получает все оставшиеся подстроки после третьей.

Замечание.Имена переменных i, j, k должны следовать в алфавитном порядке.

В нашем примере переменная `%%i` явно описана в инструкции `FOR`, а переменные `%%j` и `%%k` описываются **неявно** с помощью ключа `TOKENS=`.
Например, если в файле myfile.txt были записаны следующие три строки: AAA BBBB CCCC,GGGG DDDD

EEEEE,JJJ KKKK

;TTTT LLLL MMMMM

то в результате выполнения пакетного файла parser.bat на экран выведется следующее:

BBBB CCCC GGGG DDDD JJJ KKKK

Замечание. Ключ `TOKENS=` позволяет извлечь из одной строки файла до

26 подстрок, поэтому запрещено использовать имена переменных, начинающиеся не с букв английского алфавита (a–z). Следует помнить, что имена переменных `FOR` являются **глобальными**, поэтому одновременно не может быть активно более 26 переменных.

Команда `FOR /F` также позволяет обработать отдельную строку. Для этого следует ввести нужную строку в кавычках вместо набора имен файлов в скобках. Строка будет обработана так, как будто она взята из файла. Например, файл следующего содержания:

@ECHO OFF
FOR /F "EOL=; TOKENS=2,3* DELIMS=, " %%i IN ("AA CC BB,GG DD")
DO @ECHO %%i %%j %%k
при своем выполнении напечатает CC BB GG DD
Вместо явного задания строки для разбора можно пользоваться переменными среды, например:
@ECHO OFF
SET M=AAA БББББ BBBB,ГГГГГ ДДДД FOR /F "EOL=; TOKENS=2,3* DELIMS=,
" %%i IN ("%M%") DO @ECHO %%i %%j %%k

Наконец, команда `FOR /F` позволяет обработать **строку вывода другой команды**. Для этого следует вместо набора имен файлов в скобках ввести строку вызова команды в апострофах (не в кавычках!). Строка передается для выполнения интерпретатору команд cmd.exe, а вывод этой команды записывается в память и обрабатывается так, как будто строка вывода взята из файла. Например, следующий командный файл:

@ECHO OFF CLS
ECHO Имена переменных среды:
ECHO.
FOR /F "DELIMS==" %%i IN ('SET') DO ECHO %%i
выведет перечень имен всех переменных среды, определенных в настоящее время в системе.

В цикле `FOR` допускается применение тех же синтаксических конструкций (операторов), что и для заменяемых параметров – (табл 5).

Таблица 5 - Операторы для переменных команды FOR

Операторы	**Описание**
`%~Fi`	Переменная `%i` расширяется до полного имени файла
`%~Di`	Из переменной `%i` выделяется только имя диска
`%~Pi`	Из переменной `%i` выделяется только путь к файлу
`%~Ni`	Из переменной `%i` выделяется только имя файла
`%~Xi`	Из переменной `%i` выделяется расширение имени файла
`%~Si`	Значение операторов `N` и `X` для переменной `%i` изменяется так, что они работают с кратким именем файла
`%~Zi`	Определяется длина (размер) файла с указанным именем

Замечание. Если планируется использовать расширения подстановки значений в команде FOR, то следует внимательно подбирать имена переменных, чтобы они не пересекались с обозначениями формата.

Например, если мы находимся в каталоге C:\Program Files\Far и запустим командный файл следующего содержания:

@ECHO OFF CLS

FOR %%i IN (*.txt) DO ECHO %%~Fi

то на экран выведутся полные имена всех файлов с расширением txt:
C:\Program Files\Far\Contacts.txt

C:\Program Files\Far\ReadMe.txt C:\Program Files\Far\register.txt C:\Program Files\Far\WhatsNew.txt

Вычисление суммарной длины всех файлов в заданном подкаталоге @ECHO OFF

SETLOCAL ENABLEDELAYEDEXPANSION

Set /a Size = 0

For %%I in (%1*.*) do set /a Size= Size + %%~zI Echo %Size%

Циклы и связывание времени выполнения для переменных

Как и в рассмотренном выше примере с составными выражениями, при обработке переменных среды внутри цикла могут возникать труднообъяснимые ошибки, связанные с ранними связыванием переменных. Рассмотрим пример. Пусть имеется командный файл следующего содержания:

SET a=

FOR %%i IN (Раз,Два,Три) DO SET a=%a%%%i ECHO a=%a%

В результате его выполнения на экран будет выведена строка "a=Три", то есть фактически команда

FOR %%i IN (Раз,Два,Три) DO SET a=%a%%%i

равносильна команде

FOR %%i IN (Раз,Два,Три) DO SET a=%%i

Для исправления ситуации нужно, как и в случае с составными выражениями, вместо знаков процента (%) использовать восклицательные знаки и предварительно включить режим связывания времени выполнения командой SETLOCAL ENABLEDELAYEDEXPANSION. Таким образом, наш пример следует переписать следующим образом:

SETLOCAL ENABLEDELAYEDEXPANSION SET a=

FOR %%i IN (One,Two,Three) DO SET a=!a!%%i ECHO a=%a%

В этом случае на экран будет выведена строка "a=OneTwoThree".

Команда Findstr и ее использование в цикле

Назначение команды - поиск строк в текстовых файлах.

FINDSTR [/B] [/E] [/L] [/R] [/S] [/I] [/X] [/V] [/N] [/M] [/O] [/P] [/F:файл] [/C:строка] [/G:файл] [/D:список_папок] [/A:цвета] [/OFF[LINE]] строки

[[диск:][путь]имя_файла[...]]

/L-Поиск строк дословно.
/R-Поиск строк как регулярных выражений.
/S-Поиск файлов в текущей папке и всех ее подпапках.
/I-Определяет, что поиск будет вестись без учета регистра.
/X-Печатает строки, которые совпадают точно.
/V-Печатает строки, не содержащие совпадений с искомыми.
/N-Печатает номер строки, в которой найдено совпадение, и ее содержимое.
/M-Печатает только имя файла, в которой найдено совпадение.
/O-Печатает найденный строки через пустую строку.
/P-Пропускает строки, содержащие непечатаемые символы.
/F:файл-Читает список файлов из заданного файла (/ для консоли).
/C:строка-Использует заданную строку как искомую фразу поиска.
/D:список_папок-Поиск в списке папок (разделяются точкой с запятой). строка Искомый текст.
[диск:][путь]имя_файла - задает имя файла или файлов.

Использовать пробелы для разделения нескольких искомых строк, если аргумент не имеет префикса /C. Например, 'FINDSTR "Привет мир" a.b' ищет "Привет" или "мир" в файле a.b, а команда 'FINDSTR /C:"Привет мир" a.b' ищет строку
"Привет мир" в файле a.b.
Краткая сводка по синтаксису регулярных выражений:
. Любой символ.
* Повтор: ноль или более вхождений предыдущего символа или класса
^ Позиция в строке: начало строки
$ Позиция в строке: конец строки
[класс] Класс символов: любой единичный символ из множества [^класс] Обратный класс символов: любой единичный символ из дополнения
[x-y] Диапазон: любые символы из указанного диапазона
\x Служебный символ: символьное обозначение служебного символа x
\<xyzПозиция в слове: в начале слова xyz\> Позиция в слове: в конце слова

Пример командного файла для поиска в файле num.txt по образцу строк, в которых присутствует хотя бы одна двоичная цифра.

```
@echo off set /a kol=0
for /f %%b in ('findstr /rc:"[0-1]" num.txt') do set /a kol=kol+1 echo %kol%
```

МЕТОДИКА ВЫПОЛНЕНИЯ

1. Ознакомиться с теоретическими сведениями.
2. Запустить интерпретатор командной строки

3. Увеличить размер окна интерпретатора и задать цвет фона и цвет шрифта (рекомендуется синий фон и белый шрифт).

4. Создать список фамилий студентов группы, используя пример 1. Отсортировать список в алфавитном порядке и сохранить его в новом файле.

Замечание 1. При создании текстового файла интерпретатор командной строки использует кодировку **кириллица (DOS)**. Поэтому рекомендуется переназначить вывод в файл с расширением .**txt,** а для просмотра содержимого файла использовать Internet Explorer, указав вид кодировки кириллица (DOS).

Замечание 2. Интерпретатор хранит историю введенных команд в буфере (размером 50 строк). Для просмотра содержимого буфера используйте клавиши клавиатуры СТРЕЛКА ВВЕРХ и СТРЕЛКА ВНИЗ. Полученную команду можно отредактировать и выполнить снова.

5. Создать текстовый файл, содержащий справочные сведения по командам DIR, COPY и XCOPY.

6. Вывести содержимое указанного в табл.6 каталога по указанному формату на экран и в файл.

Таблица 6 – Варианты заданий для студентов по списку

Номера бригад	Имя каталога	Что выводить	Сортировать по	Атрибуты фай-лов и каталогов
1, 6	%Windows%	Только файлы	По размеру	Системный
2, 7	%Windows%	Файлы и подкаталоги	По дате	Скрытый
3, 8	%Windows%	Только подкаталоги	Именам	Только чтение
4, 9	%Windows% и все подкаталоги	Только файлы bmp	По размеру	Только чтение
5, 10	%Windows% и все подкаталоги	Только файлы jpg	Именам	Любые

7. Скопировать все имеющиеся в каталоге Windows растровые графические файлы в каталог WinGrafika на диске C:. Если диск C: недоступен, использовать любой другой доступный диск. / Скопировать все имеющиеся в каталоге Windows исполняемые файлы в каталог WinEx на диске C:. Если диск C: недоступен, использовать любой другой доступный диск.

Замечание. Для подготовки текстов командных файлов рекомендуется использовать блокнот (Notepad). При этом следует избегать использования в выводимых на экран результатах работы командного файла букв русского алфавита.

8. Разработать и выполнить командные файлы (КФ), выполняющие следующие функции:

1) Вывод на экран имен всех файлов с указанным расширением, находящихся в

каталоге, имя которого задается при запуске командного файла первым параметром. Расширение файлов задается вторым параметром.

2) Среди введенных с клавиатуры целых чисел (использовать SET /P) найти наибольшее и наименьшее. Признак конца ввода – знак -.

3) В заданном каталоге и его подкаталогах найти общее количество подкаталогов. На экран вывести только требуемый результат.

4) В каталогах, имена которых заданы первым и вторым параметрами командного файла, найти и вывести на экран имена файлов (расширения могут быть любые), присутствующие как в первом, так и во втором каталоге. Следует использовать только один оператор FOR.

5) Вычисление и вывод на экран значения факториала целого числа, задаваемого при запуске КФ. Предусмотреть проверку заданного значения и при задании отрицательного значения или значения, превышающего максимально возможную величину, выводить соответствующие сообщения. Для проверки правильности вычислений использовать калькулятор.

9. Разработать и выполнить КФ в соответствии с таблицей 7 (индивидуальные задания для студентов), номер задания соответствует списочному номеру студента.

Таблица 7 - Индивидуальные задания для студентов

Номер	Действия, выполняемые КФ
1	1. Подсчет количества целых чисел в текстовом файле. Считать, что слова в файле записаны в формате ОДНО СЛОВО В СТРОКЕ. Слово – это целое число (состоящее из десятичных цифр) или последовательность букв латинского алфавита (начинающаяся с буквы). Имя файла задается первым параметром КФ. 2. Вывод на экран списка файлов, хранящихся в указанном первым параметром каталоге и созданных в первом полугодии (месяцы 1-6) года, указанного вторым параметром КФ.
2	1.В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти файл наибольшего размера с расширением, указанным вторым параметром КФ. 2.В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти ТРИ файла самого большого размера. Вывести имена файлов, их размеры и даты создания

3	1. Разбиение текстового файла, имя которого задано первым параметром КФ, на три файла с именами 1.txt, 2.txt и 3.txt. Количество строк в каждом из этих файлов задано вторым, третьим и четвертым параметрами КФ. Проверить наличие указанного исходного файла и вывести сообщение о его отсутствии, проверить наличие остальных параметров и их значения на допустимость 2. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти суммарный объем файлов, имеющих расширение, указанное вторым параметром КФ.
4	1. Удаление из каталога, заданного первым параметром, файлов, которые присутствуют и в каталоге, указанным вторым параметром. Предусмотреть запрос пользователю на подтверждение удаления. 2. В каталоге, указанном первым параметром КФ, и его подкаталогах, найти файлы, созданные во второй половине рабочего дня (после 14 часов) и скопировать их в отдельный подкаталог.
5	1. Нахождение суммарного объема файлов с атрибутом system, хранящихся в каталоге, имя которого задано первым параметром КФ. 2. Проверить наличие файла Numb.txt в каталоге, указанном первым параметром КФ. Прочитать целые числа из файла, найти среди них простые и вывести результаты на экран. Считать, что все числа не превышают значения 2500.
6	1. Поиск на диске C: (или любом доступном диске) файла с заданным именем. Если файл не найден – вывод сообщения. Если файл найден – открыть его для редактирования. 2. Проверка наличия на диске в каталоге, указанном первым параметром КФ, файла FNames.txt, содержащего список имен файлов и подкаталогов. Если он есть – проверка наличия перечисленных в списке файлов и вывод имен отсутствующих. Если файла FNames нет, создание его и запись имен файлов и подкаталогов.
7	1. Вывод списка DLL (хранящихся на доступном диске), созданных до 12.2021 размером до 12000 байтов. 2. Проверка наличия на диске в каталоге, указанном первым параметром КФ, файла Numbers.txt, содержащего 2 столбца целых чисел, столбцы располагаются с позиций 2 и 20 и отделены пробелами. Если файла нет – вывод сообщения. Если файл есть, создать новый файл, содержащий три столбца, в третий поместить сумму чисел из двух первых столбцов.
8	1. Просмотр содержимого каталога, указанного первым параметром КФ. Необходимо: 1. создать подкаталоги с именами EXE, TXT, CMD, DOC и OTHER. 2. В каждый подкаталог скопировать файлы с соответствующими расширениями. 3. Пустые подкаталоги удалить. 2. В каталоге, указанном первым параметром КФ, (и его подкаталогах) найти файлы наибольшего и наименьшего размеров. Вывести имена файлов, их размеры и даты создания.

9	1. Проверка наличия трех текстовых файлов на диске и объединения их в один файл. 2. Подсчет количества вещественных чисел и целых чисел в текстовом файле. Вещественные и целые числа подсчитать отдельно. Считать, что слова в файле записаны в формате ОДНО СЛОВО В СТРОКЕ. Слово – это целое число (состоящее из десятичных цифр) или последовательность букв латинского алфавита (начинающаяся с буквы) или последовательность десятичных цифр с точкой (.) внутри строки. Имя файла задается первым параметром КФ.
10	1. Подсчет количества слов в текстовом файле, содержащем целые числа и слова. Считать, что слова в файле записаны в формате ОДНО СЛОВО В СТРОКЕ. Число – это целое число (состоящее из десятичных цифр). Слово - последовательность букв латинского алфавита (начинающаяся с буквы). Имя файла задается первым параметром КФ. 2. Просмотр содержимого каталога, указанного первым параметром КФ. Необходимо: создать подкаталоги с именами 1, 2, ..., 12. 2. В каждый подкаталог скопировать файлы, созданные в соответствующие месяцы. Пустые подкаталоги удалить.
11	1. Подсчет количества строк в текстовом файле, имя которого задано первым параметром КФ. Проверить наличие указанного файла и вывести сообщение о его отсутствии. 2. С помощью команды DIR вывести на экран имена файлов, находящихся в каталоге, имя которого задано первым параметром КФ. Второй и остальные параметры задают расширения файлов, имена которых выводить не следует. Рекомендуется с помощью ATRIB присвоить некоторым файлам атрибут СКРЫТЫЙ – такие файлы DIR не показывает.
12	1.Поиск текстового файла по его содержимому. Считать, что слова в текстовых файлах записаны в формате ОДНО СЛОВО В СТРОКЕ. Искомое слово задается первым параметром КФ. 2. Проверка наличия трех текстовых файлов на диске и объединения их в один файл.
13	1. Вывод на экран аргументов, с которыми КФ был запущен. Число аргументов от 4 до 11. При неверном числе аргументов ничего не выполнять, сообщить об ошибке. 2. Поиск и вывод на экран минимального и максимального значения аргумента КФ. Предполагается, что все аргументы КФ – целые положительные числа.

Примечание. Для решения задач 1, 9 и 10 рекомендуется использовать команду Findstr

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Различие между внутренними и внешними командами. Примеры внешних и внутренних команд.
2. Структура команды интерпретатора.
3. Получение информации о конкретной команде.
4. Групповые символы (шаблоны) и их использование.
5. Перенаправление ввода/вывода и конвейеризация команд.
6. Условное выполнение и группировка команд.
7. Назначение символов &, &&, || и () .
8. Команды для работы с файловой системой – названия и возможности.
9. Достоинства и недостатки команд COPY и XCOPY.
10. Назначение команды ECHO и примеры ее использования.
11. Команда DIR и ее возможности.
12. В какой кодировке интерпретатор выводит информацию и как получить читаемую твердую копию?
13. Вывод сообщений и дублирование команд.
14. Использование параметров командной строки.
15. Переменные среды, получение и изменение их значений.
16. Операции со строковыми и числовыми переменными.
17. Проверка существования заданного файла и наличия переменной среды.
18. Выполнение заданной команды для всех элементов указанного множества.
19. Выполнение заданной команды для всех подходящих имен файлов.
20. Выполнение заданной команды для всех подходящих имен каталогов.
21. Выполнение заданной команды для определенного каталога, а также всех его подкаталогов.
22. Получение последовательности чисел с заданными началом, концом и шагом приращения.
23. Чтение и обработка строк из текстового файла.
24. Команда Findstr. Назначение. Ключи. Использование регулярных выражений в команде. Задание и использование класса цифр и класса букв через диапазон.
25. Операторы перехода и вызова.
26. Какое минимальное количество строк (включая @echo off) должен иметь командный файл, выводящий на экран минимальное значения двух числовых аргументов?
27. Какое минимальное количество строк (включая @echo off) должен иметь командный файл, выводящий на экран минимальное значения трех числовых аргументов?

ЛАБОРАТОРНАЯ РАБОТА №2 (2 часа)
ОБОЛОЧКА КОМАНДНОЙ СТРОКИ WINDOWS POWERSHELL

Цель работы – знакомство с основными возможностями оболочки командной строки Windows PowerShell.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Оболочка Windows PowerShell была задумана разработчиками Microsoft как более мощная среда для написания сценариев и работы из командной строки. Разработчики PowerShell преследовали несколько целей, главная из которых – создание среды составления сценариев, которая наилучшим образом подходила бы для современных версий ОС Windows и была бы более функциональной, расширяемой и простой в использовании, чем какой-либо аналогичный продукт для любой другой ОС. В первую очередь, эта среда должна была подходить для решения задач, стоящих перед системными администраторами, а также удовлетворять требованиям разработчиков программного обеспечения, предоставляя им средства для быстрой реализации интерфейсов управления к создаваемым приложениям.

Для достижения этих целей были решены следующие задачи:

1. Обеспечение прямого доступа из командной строки к объектам COM, WMI и .NET. В новой оболочке присутствуют команды, позволяющие в интерактивном режиме работать с COM-объектами, а также с экземплярами классов, определенных в информационных схемах WMI и .NET.

2. Организация работы с произвольными источниками данных в командной строке по принципу файловой системы. Например, навигация по системному реестру или хранилищу цифровых сертификатов выполняется из командной строки с помощью аналога команды CD интерпретатора Cmd.exe.

3. Разработка интуитивно понятной унифицированной структуры встроенных команд, основанной на их функциональном назначении. В новой оболочке имена всех внутренних команд (в PowerShell они называются командлетами) соответствуют шаблону "глагол- существительное", например, Get-Process (получить информацию о процессе), Stop-Service (остановить службу), Clear-Host (очистить экран консоли) и т.д. Для одинаковых параметров внутренних команд используются стандартные имена, структура параметров во всех командах идентична, все команды обрабатываются одним синтаксическим анализатором. В результате облегчается запоминание и изучение команд.

4. Обеспечение возможности расширения встроенного набора команд. Внутренние команды PowerShell могут дополняться командами, создаваемыми пользователем. При этом они полностью интегрируются в оболочку, информация о них может быть получена из стандартной справочной системы PowerShell.

5. Организация поддержки знакомых команд из других оболочек. В PowerShell на уровне псевдонимов собственных внутренних команд поддерживаются наиболее часто используемые стандартные команды из оболочки Cmd.exe и Unix-оболочек. Например, если пользователь, привыкший работать с Unix-оболочкой, выполнит ls,

то он получит ожидаемый результат: список файлов в текущем каталоге (то же самое относится к команде dir).

6. Разработка полноценной встроенной справочной системы для внутренних команд. Для большинства внутренних команд в справочной системе дано подробное описание и примеры использования. В любом случае встроенная справка по любой внутренней команде будет содержать краткое описание всех ее параметров.

7. Реализация автоматического завершения при вводе с клавиатуры имен команд, их параметров, а также имен файлов и папок. Данная возможность значительно упрощает и ускоряет ввод команд с клавиатуры.

Главной особенностью среды PowerShell, отличающей ее от всех других оболочек командной строки, является то, что единицей обработки и передачи информации здесь является **объект**, а не строка текста.

При разработке любого языка программирования одним из основных является вопрос о том, какие типы данных и каким образом будут в нем представлены. При создании PowerShell разработчики решили не изобретать ничего нового и воспользоваться унифицированной объектной моделью .NET.

Рассмотрим пример. В Windows есть консольная утилита tasklist.exe, которая выдает информацию о процессах, запущенных в системе, рисунок 1

C:\>tasklist

```
Имя образа                     PID Имя сессии    № сеанса    Память
========================= ====== ============ ======== ============
System Idle Process              0                    0         16 КБ
System                           4                    0         32 КБ
smss.exe                       560                    0         68 КБ
csrss.exe                      628                    0      4 336 КБ
winlogon.exe                   652                    0      3 780 КБ
services.exe                   696                    0      1 380 КБ
lsass.exe                      708                    0      1 696 КБ
svchost.exe                    876                    0      1 164 КБ
svchost.exe                    944                    0      1 260 КБ
svchost.exe                   1040                    0     10 144 КБ
svchost.exe                   1076                    0        744 КБ
svchost.exe                   1204                    0        800 КБ
spoolsv.exe                   1296                    0      1 996 КБ
kavsvc.exe                    1516                    0      9 952 КБ
klnagent.exe                  1660                    0      5 304 КБ
klswd.exe                     1684                    0         64 КБ
```

Рисунок 1-Информация о процессах

Предположим, что мы в командном файле интерпретатора Cmd.exe с помощью этой утилиты хотим определить, сколько оперативной памяти тратит процесс kavsvc.exe. Для этого нужно выделить из выходного потока команды `tasklist` соответствующую строку, извлечь из нее подстроку, содержащую нужное число и убрать пробелы между разрядами. В PowerShell задача решается с помощью команды `get-process`, которая возвращает **коллекцию объектов**, каждый из которых соответствует одному запущенному процессу. Для определения памяти, затрачиваемой процессом kavsvc.exe, нет необходимости в дополнительных

манипуляциях с текстом, достаточно просто взять значение свойства WS объекта, соответствующего данному процессу.

Наконец, объектная модель .NET позволяет PowerShell напрямую использовать функциональность различных библиотек, являющихся частью платформы .NET. Например, чтобы узнать, каким днем недели было 9 ноября 2020 года, в PowerShell можно выполнить следующую команду:

(get-date "09.11.2020»).dayofweek.tostring()

В этом случае команда get-date возвращает .NET-объект DateTime, имеющий свойство DayOfWeek, при обращении к которому вычисляется день недели для соответствующей даты.

1 Запуск оболочки. Выполнение команд

Для запуска оболочки следует нажать на кнопку Пуск (Start), открыть меню Все программы (All Programs), выбрать элемент Стандартные, Windows PowerShell и Windows PowerShell ISE. Другой вариант запуска оболочки – пункт Выполнить… (Run) в меню Пуск (Start), ввести имя файла powershell_ise и нажать кнопку OK.

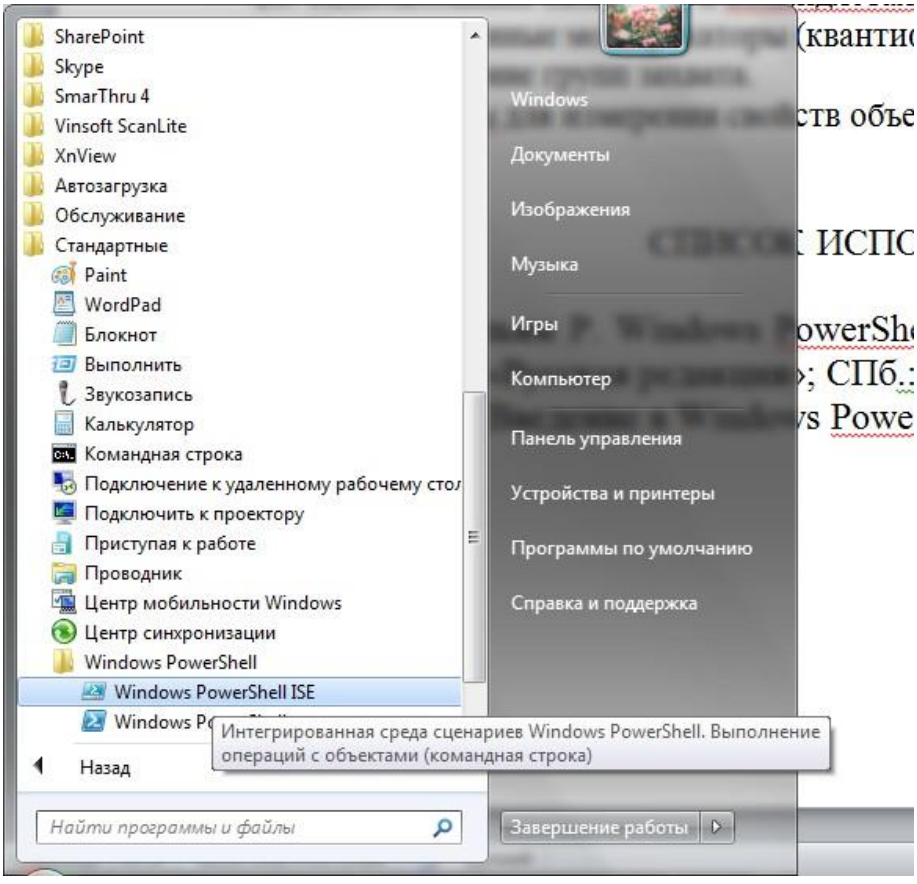

Рисунок 1 - Запуск PowerShell ISE с помощью меню

В результате откроется новое командное окно с приглашением вводить команды (рис. 2). В нижней части окна вводятся команды. Средняя часть окна содержит результаты выполнения введенной команды или сообщения об ошибках. Верхняя часть используется для работы с командными файлами.

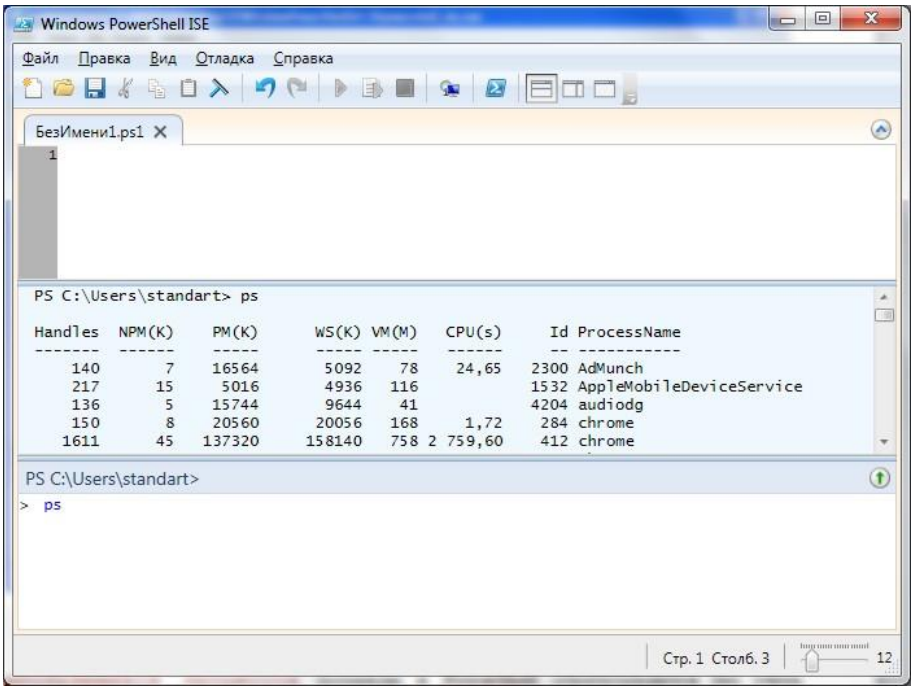

Рисунок 2 - Командное окно оболочки PowerShell ISE

Выполним первую команду в PowerShell - команду `ps – список выполняющихся процессов` (команды в PowerShell обрабатываются без учета регистра). На экран будет выведен список выполняющихся процессов.

Предыстория введенных команд работает также, как и в CMD.

2 Типы команд PowerShell

В оболочке PowerShell поддерживаются команды четырех типов: командлеты, функции, сценарии и внешние исполняемые файлы.

Первый тип – так называемые **командлеты** (cmdlet). Этот термин используется пока только внутри PowerShell. Командлет – аналог внутренней команды интерпретатора командной строки – представляет собой класс .NET, порожденный от базового класса `Cmdlet`; разрабатываются командлеты с помощью пакета PowerShell Software Developers Kit (SDK). Единый базовый класс `Cmdlet` гарантирует совместимый синтаксис всех командлетов, а также автоматизирует анализ параметров командной строки и описание синтаксиса командлетов для встроенной справки. Командлеты рассматриваются в данной работе. С командами других типов можно ознакомиться из дополнительных источников.

Данный тип команд компилируется в динамическую библиотеку (DLL) и подгружается к процессу PowerShell во время запуска оболочки (то есть сами по себе командлеты не могут быть запущены как приложения, но в них содержатся исполняемые объекты). Командлеты – это аналог внутренних команд традиционных оболочек.

Следующий тип команд – **функции**. Функция – это блок кода на языке PowerShell, имеющий название и находящийся в памяти до завершения текущего

сеанса командной оболочки. Функции, как и командлеты, поддерживают именованные параметры. Анализ синтаксиса функции производится один раз при ее объявлении.

Сценарий – это блок кода на языке PowerShell, хранящийся во внешнем файле с расширением ps1. Анализ синтаксиса сценария производится при каждом его запуске.

Последний тип команд – **внешние исполняемые файлы**, которые выполняются обычным образом операционной системой.

3 Имена и синтаксис командлетов

В PowerShell аналогом внутренних команд являются командлеты. Командлеты могут быть очень простыми или очень сложными, но каждый из них разрабатывается для решения одной, узкой задачи. Работа с командлетами становится по-настоящему эффективной при использовании их композиции (конвейеризации объектов между командлетами).

Команды Windows PowerShell следуют определенным правилам именования: Команды Windows PowerShell состоят из глагола и существительного (всегда в единственном числе), разделенных тире. Глагол задает определенное действие, а существительное определяет объект, над которым это действие будет совершено. Команды записываются на английском языке. Пример: Get-Help вызывает интерактивную справку по синтаксису Windows PowerShell.

Перед **параметрами** ставится символ «-». Например: Get-Help – Detailed.

В Windows PowerShell также включены псевдонимы многих известных команд. Это упрощает знакомство и использование Windows PowerShell. Пример: команды help (классический стиль Windows) и man (классический стиль Unix) работают так же, как и Get-Help.

Например, `Get-Process` (получить информацию о процессе), `Stop-Service` (остановить службу), `Clear-Host` (очистить экран консоли) и т.д. Чтобы просмотреть список командлетов, доступных в ходе текущего сеанса, нужно выполнить командлет `Get-Command`.

По умолчанию командлет `Get-Command` выводит сведения в трех столбцах: `CommandType`, `Name` и `Definition`. При этом в столбце `Definition` отображается синтаксис командлетов (многоточие (…) в столбце синтаксиса указывает на то, что данные обрезаны).

Замечание. Косые черты (`/` и `\`) вместе с параметрами в оболочке Windows PowerShell не используются.

В общем случае синтаксис командлетов имеет следующую структуру:

имя_командлета –параметр1 –параметр2 аргумент1 аргумент2

Здесь `параметр1` – параметр (переключатель), не имеющий значения; `параметр2` – имя параметра, имеющего значение `аргумент1`; `аргумент2` – параметр, не имеющий имени. Например, командлет `Get- Process` имеет параметр `Name`, который определяет имя процесса, информацию о котором нужно вывести. Имя этого параметра указывать необязательно. Таким образом, для

получения сведений о процессе `Far` можно ввести либо команду `Get-Process -Name Far`, либо команду `Get-Process Far`.

4 Автоматическое завершение команд (автозавершение ввода команд)

Находясь в оболочке PowerShell, можно ввести часть какой-либо команды, нажать клавишу <Tab> и система попытается сама завершить ввод этой команды.

Подобное автоматическое завершение срабатывает, во-первых, для имен файлов и путей файловой системы. При нажатии клавиши <Tab> PowerShell автоматически расширит частично введенный путь файловой системы до первого найденного совпадения. При повторении нажатия клавиши <Tab> производится циклический переход по имеющимся возможностям выбора. Также в PowerShell реализована возможность автоматического завершения путей файловой системы на основе шаблонных символов (`?` И `*`). Например, если ввести команду `cd c:\pro*files` и нажать клавишу <Tab>, то в строке ввода появится команда `cd 'C:\Program Files'`.

Во-вторых, в PowerShell реализовано автозавершение имен командлетов и их параметров. Если ввести первую часть имени командлета (глагол) и дефис, нажать после этого клавишу <Tab>, то система подставит имя первого подходящего командлета (следующий подходящий вариант имени выбирается путем повторного нажатия <Tab>). Аналогичным образом автозавершение срабатывает для частично введенных имен параметров командлета: нажимая клавишу <Tab>, мы будем циклически перебирать подходящие имена.

Наконец, PowerShell позволяет автоматически завершать имена используемых переменных (объектов) и имена свойств объектов.

5 Псевдонимы команд

Механизм псевдонимов, реализованный в оболочке PowerShell, дает возможность пользователям выполнять команды по их **альтернативным именам** (например, вместо команды `Get-Childitem` можно пользоваться псевдонимом `dir`). В PowerShell заранее определено много псевдонимов, можно также добавлять собственные псевдонимы в систему.

Псевдонимы в PowerShell делятся на два типа. Первый тип предназначен для совместимости имен с разными интерфейсами. Псевдонимы этого типа позволяют пользователям, имеющим опыт работы с другими оболочками (Cmd.exe или Unix-оболочки), использовать знакомые им имена команд для выполнения аналогичных операций в PowerShell, что упрощает освоение новой оболочки, позволяя не тратить усилий на запоминание новых команд PowerShell. Например, пользователь хочет очистить экран. Если у него есть опыт работы с Cmd.exe, то он, естественно, попробует выполнить команду `cls`. PowerShell при этом выполнит командлет `Clear-Host`, для которого cls является псевдонимом и который выполняет требуемое действие – очистку экрана. Для пользователей Cmd.exe в PowerShell определены псевдонимы `cd, cls, copy, del, dir, echo, erase, move,`

popd, pushd, ren, rmdir, sort, type; для пользователей Unix – псевдонимы cat, chdir, clear, diff, h, history, kill, lp, ls, mount, ps, pwd, r, rm, sleep, tee, write.

Узнать, какой именно командлет скрывается за знакомым псевдонимом, можно с помощью командлета Get-Alias:

```
PS C:\> Get-Alias cd
CommandType     Name      Definition
-----------     ----      ----------
Alias           cd        Set-Location
```

Псевдонимы второго типа (стандартные псевдонимы) в PowerShell предназначены для быстрого ввода команд. Такие псевдонимы образуются из имен командлетов, которым они соответствуют. Например, глагол Get сокращается до g, глагол Set сокращается до s, существительное Location сокращается до l и т.д. Таким образом, для командлету Set- Location соответствует псевдоним sl, а командлету Get-Location – псевдоним gl.

Просмотреть список всех пседонимов, объявленных в системе,

можно с помощью командлета Get-Alias без параметров. Определить собственный псевдоним можно с помощью командлета Set-Alias.

6 Справочная система PowerShell

В PowerShell предусмотрено несколько способов получения справочной информации внутри оболочки.

Краткую справку по одному командлету можно получить с помощью параметра ? (вопросительный знак), указанного после имени этого командлета. Например:

```
PS C:\> get-process -?
```

Вместо help или man в Windows PowerShell можно также использовать команду Get-Help. Ее синтаксис описан ниже:

Get-Help выводит на экран справку об использовании справки Get-Help *
перечисляет все команды Windows PowerShell

Get-Help *команда* выводит справку по соответствующей команде

Get-Help команда –Detailed выводит подробную справку с примерами
команды

Использование команды help для получения подробных сведений о команде help:

```
Get-Help
Get-Help –Detailed.
```

Комана Get-Help позволяет просматривать справочную информацию не только о разных командлетах, но и о синтаксисе языка PowerShell, о псевдонимах и т. д.

Например, чтобы прочитать справочную информацию об использовании массивов в PowerShell, нужно выполнить следующую команду: Get-Help about_array.

Командлет Get-Help выводит содержимое раздела справки на экран сразу

целиком. Функции `man` и `help` позволяют справочную информацию выводить поэкранно (аналогично команде `MORE` интерпретатора Cmd.exe), например: `man about_array`.

7 Конвейеризация и управление выводом команд Windows PowerShell

Ранее было рассмотрено понятие конвейеризации (или композиции) команд интерпретатора Cmd.exe, когда выходной поток одной команды перенаправлялся во входной поток другой, объединяя тем самым две команды вместе. Подобные конвейеры команд используются в большинстве оболочек командной строки и являются средством, позволяющим передавать информацию между разными процессами. Механизм композиции команд представляет собой, вероятно, наиболее ценную концепцию, используемую в интерфейсах командной строки. Конвейеры не только снижают усилия, прилагаемые при вводе сложных команд, но и облегчают отслеживание потока работы в командах.

В оболочке PowerShell также очень широко используется механизм конвейеризации команд, однако здесь по конвейеру передается не поток текста, как во всех других оболочках, а объекты. При этом с элементами конвейера можно производить различные манипуляции: фильтровать объекты по определенному критерию, сортировать и группировать объекты, изменять их структуру (ниже мы подробнее рассмотрим операции фильтрации и сортировки элементов конвейера).

7.1 Конвейеризация объектов в PowerShell

Конвейер в PowerShell – это последовательность команд, разделенных между собой знаком `|` (вертикальная черта). Каждая команда в конвейере получает объект от предыдущей команды, выполняет определенные операции над ним и передает следующей команде в конвейере. С точки зрения пользователя, объекты упаковывают связанную информацию в форму, в которой информацией проще манипулировать как единым блоком и из которой при необходимости извлекаются определенные элементы.

Передача данных между командами в виде объектов имеет большое преимущество над обычным обменом информацией посредством потока текста. Ведь команда, принимающая поток текста от другой утилиты, должна его проанализировать, разобрать и выделить нужную ей информацию, а это может быть непросто, так как обычно вывод команды больше ориентирован на визуальное восприятие человеком (это естественно для интерактивного режима работы), а не на удобство последующего синтаксического разбора.

При передаче по конвейеру объектов этой проблемы не возникает, здесь нужная информация извлекается из элемента конвейера простым обращением к соответствующему свойству объекта. Однако возникает новый вопрос: каким образом узнать, какие именно свойства есть у объектов, передаваемых по конвейеру? Ведь при выполнении того или иного командлета мы на экране видим только одну или несколько колонок отформатированного текста.

Пример Запустим командлет Get-Process, который выводит информацию о

запущенных в системе процессах (рис.3):

PS C:\> Get-Process

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
158	11	45644	22084	126	159.69	2072	AcroRd32
98	5	1104	284	32	0.10	256	alg
39	1	364	364	17	0.26	1632	ati2evxx
57	3	1028	328	30	0.38	804	atiptaxx
434	6	2548	3680	27	21.96	800	csrss
64	3	812	604	29	0.22	1056	ctfmon
364	11	14120	9544	69	11.82	456	explorer
24	2	1532	2040	29	5.34	2532	Far

Рисунок 3 - Информацию о запущенных в системе

Фактически на экране мы видим только сводную информацию (результат форматирования полученных данных), а не полное представление выходного объекта. Из этой информации непонятно, сколько точно свойств имеется у объектов, генерируемых командой `Get- Process`, и какие имена имеют эти свойства. Например, мы хотим найти все "зависшие" процессы, которые не отвечают на запросы системы. Можно ли это сделать с помощью командлета `Get-Process`, какое свойство нужно проверять у выводимых объектов?

Для ответа на подобные вопросы нужно научиться исследовать структуру объектов PowerShell, узнавать, какие свойства и методы имеются у этих объектов.

7.2 Просмотр структуры объектов

Для анализа структуры объекта, возвращаемого определенной командой, проще всего направить этот объект по конвейеру на командлет `Get-Member` (псевдоним `gm`), например (рис.4):

PS C:\> Get-Process | Get-Member

```
   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize
PM                         AliasProperty  PM = PagedMemorySize
VM                         AliasProperty  VM = VirtualMemorySize
WS                         AliasProperty  WS = WorkingSet
. . .
Responding                 Property       System.Boolean Responding {get;}
. . .
```

Рисунок 4 - объект по конвейеру на командлет `Get-Member`

Здесь мы видим имя .NET-класса, экземпляры которого возвращаются в ходе работы исследуемого командлета (в нашем примере это класс System.Diagnostic.Process), а также полный список элементов объекта (в частности, интересующее нас свойство Responding, определяющего "зависшие" процессы). При

этом на экран выводится очень много элементов, просматривать их неудобно. Командлет `Get-Member` позволяет перечислить только те элементы объекта, которые являются его **свойствами**. Для этого используется параметр `MemberType` со значением `Properties`:(рис.5)

PS C:\> Get-Process | Get-Member -MemberType Property

```
   TypeName: System.Diagnostics.Process
Name                       MemberType Definition
----                       ---------- ----------
BasePriority               Property   System.Int32 BasePriority {get;}
EnableRaisingEvents        Property   System.Boolean EnableRaisingEvents...
ExitCode                   Property   System.Int32 ExitCode {get;}
ExitTime                   Property   System.DateTime ExitTime {get;}
Handle                     Property   System.IntPtr Handle {get;}
HandleCount                Property   System.Int32 HandleCount {get;}
HasExited                  Property   System.Boolean HasExited {get;}
Id                         Property   System.Int32 Id {get;}
. . .
Responding                 Property   System.Boolean Responding {get;}
. . .
```

Рисунок 5 - Элементы объекта, которые являются его свойствами

Процессам ОС соответствуют объекты, имеющие очень много свойств, на экран же при работе командлета `Get-Process` выводятся лишь несколько из них (способы отображения объектов различных типов задаются конфигурационными файлами в формате XML, находящимися в каталоге, где установлен файл powershell.exe).

Рассмотрим наиболее часто используемые операции над элементами конвейера: фильтрации и сортировки.

7.3 Фильтрация объектов в конвейере

В PowerShell поддерживается возможность **фильтрации объектов** в конвейере, т.е. удаление из конвейера объектов, не удовлетворяющих определенному условию. Данную функциональность обеспечивает командлет `Where-Object`, позволяющий проверить каждый объект, находящийся в конвейере, и передать его дальше по конвейеру, только если объект удовлетворяет условиям проверки.

Например, для вывода информации о "зависших" процессах (объекты, возвращаемые командлетом `Get-Process`, у которых свойство `Responding` равно `False`) можно использовать следующий конвейер: Get-Process | Where-Object {-not $_.Responding}

Другой пример – оставим в конвейере только те процессы, у которых значение идентификатора (свойство `Id`) больше 1000:

Get-Process | Where-Object {$_.Id -gt 1000}

В блоках сценариев командлета `Where-Object` для обращения к текущему объекту конвейера и извлечения нужных свойств этого объекта используется **специальная переменная** `$_`, которая создается оболочкой PowerShell

автоматически. Данная переменная используется и в других командлетах, производящих обработку элементов конвейера.

Условие проверки в `Where-Object` задается в виде **блока сценария** – одной или нескольких команд PowerShell, заключенных в фигурные скобки `{}`. Результатом выполнения данного блока сценария должно быть значение логического типа: `True` (истина) или `False` (ложь). Как можно понять из примеров, в блоке сценария используются специальные операторы сравнения.

Замечание. В PowerShell для операторов сравнения не используются обычные символы `>` или `<`, так как в командной строке они обычно означают перенаправление ввода/вывода.

Основные операторы сравнения приведены в (табл. 1).

Таблица 1 - Операторы сравнения в PowerShell

Оператор	Значение	Пример (возвращается значение True)
-eq	равно	10 -eq 10
-ne	не равно	9 -ne 10
-lt	меньше	3 -lt 4
-le	меньше или равно	3 –le 4
-gt	больше	4 -gt 3
-ge	больше или равно	4 -ge 3
-like	сравнение на совпадение с учетом подстановочного знака в тексте	"file.doc" –like "f*.doc"
-notlike	сравнение на несовпадение с учетом подстановочного знака в тексте	"file.doc" –notlike "f*.rtf"
-contains	содержит	1,2,3 –contains 1
-notcontains	не содержит	1,2,3 –notcontains 4

Операторы сравнения можно соединять друг с другом с помощью логических операторов (см. табл. 2).

Таблица 2 - Логические операторы в PowerShell

Оператор	Значение	Пример (возвращается значение True)
-and	логическое И	(10 -eq 10) –and (1 –eq 1)
-or	логическое ИЛИ	(9 -ne 10) –or (3 –eq 4)
-not	логическое НЕ	-not (3 –gt 4)
!	логическое НЕ	!(3 -gt 4)

7.4 Сортировка объектов

Сортировка элементов конвейера – еще одна операция, которая часто применяется при конвейерной обработке объектов. Данную операцию осуществляет командлет `Sort-Object`: ему передаются имена свойств, по которым нужно произвести сортировку, а он возвращает данные, упорядоченные по значениям этих свойств.

Например, для вывода списка запущенных в системе процессов, упорядоченного по затраченному процессорному времени (свойство `cpu`), можно воспользоваться следующим конвейером:

PS C:\> Get-Process | Sort-Object cpu

Для сортировки в обратном порядке используется параметр `Descending`: PS C:\> Get-Process | Sort-Object cpu -Descending

В рассмотренных нами примерах конвейеры состояли из двух командлетов. Это не обязательное условие, конвейер может объединять и большее количество команд, например:

Get-Process | Where-Object {$_.Id -gt 1000} | Sort-Object cpu –Descending

7.5 Использование переменных

В переменных хранятся все возможные значения, даже если они являются объектами. Имена переменных в PowerShell всегда должны начинаться с символа «$». Можно сохранить список процессов в переменной, это позволит в любое время получать доступ к списку процессов. Присвоить значение переменной легко:

$a = get-process | sort-object CPU

Вывести содержимое переменной можно, просто напечатав в командной строке $a.

7.6 Создание и использование массивов

Для создания и инициализации массива достаточно присвоить значения его элементам. Значения, добавляемые в массив, разделяются запятыми и отделяются от имени массива символом присваивания. Например, следующая команда создаст массив $a из трех элементов:

PS C:\> $a=1,5,7
PS C:\>$a 1
5
7

Можно создать и инициализировать массив, используя оператор диапазона (..). Например, команда

PS C:\> $b=10..15

создает и инициализирует массив $b, содержащий 6 значений 10, 11, 12, 13, 14 и 15.

Для создания массива может использоваться операция ввода значений его элементов из текстового файла:

PS C:\> $f = Get-Content c:\data\numb.txt –TotalCount 25 PS C:\>$f.length

25

В приведенном примере результат выполнения командлета Get- Content присваивается массиву $f. Необязательный параметр –TotalCount ограничивает количество прочитанных элементов величиной 25. Свойство объекта массив – length – имеет значение, равное количеству элементов массива, в примере оно равно 25 (предполагается, что в текстовом файле munb.txt по крайней мере 25 строк).

Обращение к элементам массива

Длина массива (количество элементов) хранится в свойстве Length. Для обращения к определенному элементу массива нужно указать его индекс в квадратных скобках после имени переменной. Нумерация элементов массива **всегда начинается с нуля**. В качестве индекса можно указывать и отрицательные значения, отсчет будет вестись с конца массива

– индекс -1 соответствует последнему элементу массива.

Операции с массивами

По умолчанию массивы PowerShell могут содержать элементы разных типов (целые 32-х разрядные числа, строки, вещественные и другие), то есть являются полиморфными. Можно создать массив с жестко заданным типом, содержащий элементы только одного типа, указав нужный тип в квадратных скобках перед именем переменной. Например, следующая команда создаст массив 32-х разрядных целых чисел:

```
PS C:\> [int[]]$a=1,2,3
```

Массивы PowerShell базируются на .NET-массивах, имеющих фиксированную длину, поэтому обращение за предел массива фиксируется как ошибка. Имеется способ увеличения первоначально определенной длины массива. Для этого можно воспользоваться оператором конкатенации + или +=. Например, следующая команда добавит к массиву

$a два новых элемента со значениями 5 и 6:

```
PS C:\> $a 1
2
3
4
PS C:\>$a+=5,6
PS C:\>$a 1
2
3
4
5
6
```

При выполнении оператора += происходит следующее:

создается новый массив, размер которого достаточен для помещения в него всех элементов;

первоначальное содержимое массива копируется в новый массив; новые

элементы копируются в конец нового массива.

Таким образом, на самом деле создается новый массив большего размера.

Можно объединить два массива, например $b и $c в один с помощью операции конкатенации +. Например:

PS C:\> $d=$b+$c

8 Регулярные выражения – назначение и использование

Регулярные выражения (или сокращенно “регэкспы” (regexp, regular expressions)) обладают огромной мощью, и способны сильно упростить жизнь системного администратора или программиста. В PowerShell регулярные выражения легко доступны, удобны в использовании и максимально функциональны. PowerShell использует реализацию регулярных выражений .NET.

Регулярные выражения - это специальный мини-язык, служащий для разбора (parsing) текстовых данных. С его помощью можно разделять строки на компоненты, выбирать нужные части строк для дальнейшей обработки, производить замены и т. д.

Знакомство с регулярными выражениями начнем с более простой технологии, служащей подобным целям - с **подстановочных символов**. Наверняка вы не раз выполняли команду dir, указывая ей в качестве аргумента маску файла, например *.exe. В данном случае звёздочка означает “любое количество любых символов”. Аналогично можно использовать и знак вопроса, он будет означать “один любой символ”, то есть dir ??.exe выведет все файлы с расширением .exe и именем из двух символов. В PowerShell можно применять и еще одну конструкцию – **группы символов**. Так например [a-f] будет означать “один любой символ от a до f, то есть (a,b,c,d,e,f)”, а [smw] любую из трех букв (s, m или w). Таким образом команда get-childitem [smw]??.exe выведет файлы с расширением .exe, у которых имя состоит из трех букв, и первая буква либо s, либо m, либо w.

8.1 Оператор PowerShell -match

Для начала изучения мы будем использовать оператор PowerShell - match, который позволяет сравнивать текст слева от него, с регулярным выражением справа. В случае если текст подпадает под регулярное выражение, оператор выдаёт True, иначе – False.

PS C:\> "PowerShell" -match "Power" True

При сравнении с регулярным выражением ищется лишь вхождение строки, полное совпадение текста необязательно (разумеется, это можно изменить). То есть достаточно, чтобы регулярное выражение встречалось в тексте.

PS C:\> "Shell" -match "Power" False

PS C:\> "PowerShell" -match "rsh" True

Еще одна тонкость: оператор -match по умолчанию не чувствителен к регистру символов (как и другие текстовые операторы в PowerShell), если же нужна чувствительность к регистру, используется -cmatch:

PS C:\> "PowerShell" -cmatch "rsh" False

8.2 Использование групп символов

В регулярных выражениях можно использовать и группы символов: PS C:\> Get-Process | where {$_.name -match "sy[ns]"} (рис.6)

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
165	11	2524	8140	79	0,30	5228	mobsync
114	10	3436	3028	83	50,14	3404	SynTPEnh
149	11	2356	492	93	0,06	1592	SynTPStart
810	0	116	380	6		4	System

Рисунок 6 - Использование групп символов

И диапазоны в этих группах:

PS C:\> "яблоко","апельсин","груша","абрикос" -match "а[а-п]" апельсин абрикос

В левой части оператора -match находится массив строк, и оператор соответственно вывел лишь те строки, которые подошли под регулярное выражение.

Перечисления символов можно комбинировать, например группа [агдэ-я] будет означать “А или Г или Д или любой символ от Э до Я включительно”. Но гораздо интереснее использовать диапазоны для определения целых **классов символов**. Например [а-я] будет означать любую букву русского алфавита, а [a-z] английского. Аналогично можно поступать с цифрами – следующая команда выведет все процессы, в именах которых встречаются цифры:

PS C:\> Get-Process | where {$_.name -match "[0-9]"} (рис.7)

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
57	2	404	1620	16	0,05	984	ati2evxx
110	4	2540	4868	36	0,20	852	hpgs2wnd
105	3	940	3292	36	0,19	2424	hpgs2wnf
91	3	2116	3252	34	0,06	236	rundll32

Рисунок 7 - Процессы, в именах которых встречаются цифры

Так как эта группа используется достаточно часто, для неё была выделена специальная последовательность – \d (от слова digit). По смыслу она полностью идентична [0-9], но короче.

PS C:\> Get-Process | where {$_.name -match "\d"} (рис.8)

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
93	10	1788	2336	70	1,25	548	FlashUtil10c
158	12	6500	1024	96	0,14	3336	smax4pnp
30	6	764	160	41	0,02	3920	TabTip32

Рисунок 8 - Последовательность – \d (от слова digit)

Так же последовательность была выделена для группы “любые буквы любого алфавита, любые цифры, или символ подчеркивания” эта группа обозначается как \w (от word) она примерно эквивалентна конструкции [a-za-я_0-9] (в \w еще входят символы других алфавитов которые используются для написания слов).

Другая популярная группа: \s – “пробел, или другой пробельный символ” (например символ табуляции). Сокращение от слова space. В большинстве случаев вы можете обозначать пробел просто как пробел, но эта конструкция добавляет читабельности регулярному выражению.

Не менее популярной группой можно назвать символ . (точка). Точка в регулярных выражениях аналогична по смыслу знаку вопроса в подстановочных символах, то есть обозначает один любой символ.

Все вышеперечисленные конструкции можно использовать как отдельно, так и в составе групп, например [\s\d] будет соответствовать любой цифре или пробелу. Если вы хотите указать внутри группы символ - (тире/минус) то надо либо экранировать его символом \ (обратный слеш), либо поставить его в начале группы, чтобы он не был случайно истолкован как диапазон:

```
PS C:\> "?????","Word","123","-" -match "[-\d]" 123
-
```

8.3 Отрицательные группы и якоря

Рассмотрим некоторые более “продвинутые” конструкции регулярных выражений.

Предполагается, что вы уже знаете, как указать регулярному выражению, какие символы и/или их последовательности должны быть в строке для совпадения. А что если нужно указать не те символы, которые должны присутствовать, а те, которых не должно быть? То есть если нужно вывести лишь согласные буквы, вы можете их перечислить, а можете использовать и отрицательную группу с гласными, например:

```
PS C:\> "a","b","c","d","e","f","g","h" -match "[^aoueyi]" b
c d f g h
```

"Крышка" в качестве первого символа группы символов означает именно **отрицание**. То есть на месте группы может присутствовать любой символ кроме перечисленных в ней. Для того чтобы включить отрицание в символьных группах (\d, \w, \s), не обязательно заключать их в квадратные скобки, достаточно перевести их в **верхний регистр**. Например \D будет означать "что угодно, кроме цифр", а \S "всё кроме пробелов"

```
PS C:\> "a","b","1","c","45" -match "\D"
a b c
PS C:\> "a","-","*","c","&" -match "\W"
-
* &
```

Символьные группы позволяют указать лишь содержимое одной позиции, один символ, находящийся в неопределенном месте строки. А что если надо например выбрать все слова которые начинаются с буквы w? Если просто поместить эту букву в регулярное выражение, то оно совпадёт для всех строк, где w вообще

встречается, и не важно – в начале, в середине или в конце строки. В таких случаях на помощь приходят "**якоря**". Они позволяют производить сравнение, начиная с определенной позиции в строке.

^ (крышка) является **якорем начала строки**, а $ (знак доллара) - обозначает конец строки.

Не запутайтесь - ^ как символ отрицания используется лишь в начале группы символов, а вне группы - этот символ является уже якорем. Авторам регулярных выражений явно не хватало специальных символов, и они по возможности использовали их более чем в одном месте.

Пример1. Вывод списка процессов, имена которых начинаются с буквы w: PS C:\> Get-Process | where {$_.name -match "^w"} (рис.8)

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
80	10	1460	156	47	0,11	452	wininit
114	9	2732	1428	55	0,56	3508	winlogon
162	11	3660	1652	44	0,14	3620	wisptis
225	20	5076	4308	95	31,33	3800	wisptis

Рисунок 8 - Вывод списка процессов

Эта команда вывела процессы, у которых сразу после начала имени (^) следует символ w. Иначе говоря, имя начинается на w. Для усложнения примера, и для упрощения понимания, добавим сюда “крышку” в значении отрицательной группы:

PS C:\> Get-Process | where {$_.name -match "^w[^l-z]"} (рис.9)

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
80	10	1460	156	47	0,11	452	wininit
114	9	2732	1428	55	0,56	3508	winlogon
162	11	3660	1652	44	0,14	3620	wisptis
225	20	5076	4308	95	31,50	3800	wisptis

Рисунок 9-(^) следует символ w

Теперь команда вывела процессы, у которых имя начинается с символа w, а следующий символ является чем угодно, только не символом из диапазона l-z.

Для закрепления опробуем второй якорь – конец строки:

PS C:\> "Яблоки","Груши","Дыня","Енот","Апельсины","Персик" -match "[ыи]$"

Яблоки Груши Апельсины

Это выражение вывело нам все слова в которых последняя буква И или Ы.

Если вы можете точно описать содержимое всей строки, то вы можете использовать и оба якоря одновременно:

PS C:\> "abc","adc","aef","bca","aeb","abec","abce" -match "^a.[cb]$" abc adc aeb

Это регулярное выражение выводит все строки, которые начинаются с буквы A, за которой следует один любой символ (точка), затем символ C или B и затем конец

строки.

Обозначения некоторых классов символов (метасимволы) приведены в (табл. 2).

Таблица 2 - Метасимволы, используемые в регулярных выражениях

Метасимвол	Описание метасимвола
.(точка)	Предполагает, что в конечном выражении на ее месте будет стоять любой символ. Продемонстрируем это на примере набора английских слов: **Исходный набор строк:** wake make machine cake maze **Регулярное выражение:** ma.e **Результат:** make maze
\w	Замещает любые символы, которые относятся к буквам, цифрам и знаку подчеркивания. Пример: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\wc **Результат:** abc a1c
\W	Замещает все символы, кроме букв, цифр и знака подчеркивания (то есть является обратным метасимволу \w). Пример: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\Wc **Результат:** a$c a c
\d	Замещает все цифры. Продемонстрируем его действие на том же примере: **Исходный набор строк:** abc a$c a1c a c **Регулярное выражение:** a\dc **Результат:** alc

\D	Замещает все символы, кроме цифр, например: **Исходный набор строк:** abc a$c alc a c **Регулярное выражение:** a\Dc **Результат:** abc a$c a c

8.4 Количественные модификаторы (квантификаторы)

Обычно регулярные выражения гораздо сложнее, чем приведенные выше, и записывать их по одному символу было бы тяжеловато. Например, нужно отобрать строки, состоящие из четырех символов, каждый из которых может быть буквой от A до F или цифрой? Регулярное выражение могло бы выглядеть примерно так:

```
PS C:\> "af12","1FE0","1fz1","B009","C1212" -match "^[a-f\d][a-f\d][a-f\d][a-f\d]$"
af12 1FE0 B009
```

Не слишком то лаконично, не правда ли? К счастью всю эту конструкцию можно значительно сократить. Для этого в регулярных выражениях существует специальная конструкция – "**количественные модификаторы**" (квантификаторы). Эти модификаторы приписываются к любой группе справа, и определяют количество вхождений этой группы. Например, количественный модификатор {4} означает 4 вхождения. Посмотрим на приведенном выше примере:

```
PS C:\> "af12","1FE0","1fz1","B009","C1212" -match "^[a-f\d]{4}$"
af12 1FE0 B009
```

Данное регулярное выражение полностью эквивалентно предыдущему – "4 раза по [a-f\d]". Но этот количественный модификатор не обязательно жестко оговаривает количество повторений. Например,можно задать количество как "от 4 до 6". Делается это указанием внутри фигурных скобок двух чисел через запятую – минимума и максимума:

```
PS C:\> "af12","1FE0","1fA999","B009","C1212","A00062","FF00FF9" -match "^[a-f\d]{4,6}$" af12
1FE0
1fA999 B009 C1212 A00062
```

Если максимальное количество вхождений безразлично, например, нужно указать "3 вхождения или больше", то максимум можно просто опустить (оставив запятую на месте), например "строка состоящая из 3х или более цифр":

```
PS C:\> "1","12","123","1234","12345" -match "^\d{3,}$" 123
1234
12345
```

Минимальное значение опустить нельзя, но можно просто указать единицу:

```
PS C:\> "1","12","123","1234","12345" -match "^\d{1,3}$" 1
12
123
```

Как и в случае с символьными группами, для особенно популярных значений количественных модификаторов, есть короткие псевдонимы:

+ (плюс), эквивалентен {1,} то есть, "одно или больше вхождений"

* (звездочка), то же самое что и {0,} или на русском языке – "любое количество вхождений, в том числе и 0"

? (вопросительный знак), равен {0,1} – "либо одно вхождение, либо полное отсутствие вхождений".

В регулярных выражениях, количественные модификаторы **сами по себе** использоваться не могут. Для них обязателен символ или символьная группа, которые и будут определять их смысл. Вот несколько примеров:

.+ Один или более любых символов. Аналог ?* в простых подстановках (как в cmd.exe).

Следующее выражение выбирает процессы, у которых имя "начинается с буквы S, затем следует 1 или более любых символов, затем снова буква S и сразу после неё конец строки". Иначе говоря "имена которые начинаются и заканчиваются на S":

PS C:\> Get-Process | where {$_.name -match "^s.+s$"} (рис.10)

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
257	14	6540	5220	53	5,97	508	services
30	2	424	128	5	0,08	280	smss

Рисунок 10 -"имена которые начинаются и заканчиваются на S"

\S* Любое количество символов не являющихся пробелами. Подобное выражение может совпасть и с ""(с пустой строкой), ведь под любым количеством подразумевается и ноль, то есть 0 вхождений – тоже результат.

PS C:\> "abc", "cab", "a c","ac","abdec" -match "a\S*c" abc

ac abdec

Заметьте, строка "ac" тоже совпала, хотя между буквами A и C вообще не было символов. Если заменить * на + то будет иначе:

PS C:\> "abc", "cab", "a c","ac","abdec" -match "a\S+c" abc

abdec

бобры? (Это не вопрос, а регулярное выражение). Последовательность "бобр", после которой может идти символ "ы", а может и отсутствовать:

PS C:\> "бобр","бобры","бобрята" -match "^бобры?$" Бобр бобры

8.5 Группы захвата и переменная $matches

Теперь, когда мы можем с помощью регулярных выражений описывать и проверять строки по достаточно сложным правилам, пора познакомится с другой не менее важной возможностью регулярных выражений – «группами захвата» (capture groups). Как следует из названия, группы можно использовать для группировки. К группам захвата, как и к символам и символьным группам, можно применять количественные модификаторы. Например, следующее выражение означает «Первая буква в строке – S, затем одна или больше групп, состоящих из “знака – (минус) и

любого количества цифр за ним” до конца строки»:

PS C:\> “S-1-5-21-1964843605-2840444903-4043112481” –match “^S(-\d+)+$”

True Или:

PS C:\> “Ноут”,”Ноутбук”,”Лептоп” –match “Ноут(бук)?” Ноут

Ноутбук

Эти примеры показывают, как можно использовать группы захвата для группировки, но это вовсе не главное их качество. Гораздо важнее то, что часть строки, подпавшая под подвыражение, находящееся внутри такой группы, помещается в специальную переменную – $matches.

$Matches – это массив, и в нем может находится содержимое нескольких групп. Причем под индексом 0 туда помещается вся совпавшая строка, начиная с единицы идет содержимое групп захвата. Рассмотрим пример: PS C:\> “At 17:04 Firewall service was stopped.” –match “(\d\d:\d\d) (\S+)” True

PS C:\> $matches

Name	Value
2	Firewall
1	17:04
0	17:04 Firewall

Под индексом 0 находится вся часть строки, подпавшая под регулярное выражение, под 1 находится содержимое первых скобок, и под 2 соответственно содержимое вторых скобок. К содержимому $matches можно обращаться как к элементам любого другого массива в PowerShell: PS C:\> $matches[1]

17:04

PS C:\> $matches[2] Firewall

Если в строке присутствует много групп захвата, то бывает полезно дать им имена, это сильно облегчает дальнейшую работу с полученными данными:

PS C:\> “At 17:04 Firewall service was stopped.” –match “(?<Время>\d\d:\d\d) (?<Служба>\S+)”

True

PS C:\> $matches

Name	Value
Время	17:04
Служба	Firewall
0	17:04 Firewall

PS C:\> $matches.Время 17:04

PS C:\> $matches[«Служба»] Firewall

Регулярное выражение конечно усложнилось, но зато работать с результатами гораздо приятнее. Синтаксис именования следующий: (?<Название

Группы>подвыражение)

Не перепутайте порядок, сначала следует знак вопроса. Количественные модификаторы, в том числе ? могут применяться только после группы, и следовательно в начале подвыражения – бессмысленны. Поэтому в группах знак вопроса, следующий сразу за открывающей скобкой, означает особый тип группы, в нашем примере – **именованную.**

Другой тип группы, который часто используется – **незахватывающая** группа. Она может пригодиться в тех случаях, когда не нужно захватывать содержимое группы, а надо применить её только для группировки. Например, в вышеприведённом примере с SID, такая группа была бы более уместна:

PS C:\> “S-1-5-21-1964843605-2840444903-4043112481” -match “^S(?:-\d+)+$” True

PS C:\> $matches

Name	Value
0	S-1-5-21-1964843605-2840444903-4043112481

Синтаксис такой группы: (?:подвыражение). Группы можно и вкладывать одну в другую:

PS C:\> “MAC address is ‘00-19-D2-73-77-6F’.” –match “is ‘([a-f\d]{2}(?:-[a-f\d]{2}){5})’”

True

PS C:\> $matches

Name	Value
1	00-19-D2-73-77-6F
0	is '00-19-D2-73-77-6F'

9 Управляющие инструкции

9.1 Инструкция If …ElseIf … Else

В общем случае синтаксис инструкции If имеет вид If (*условие1*)
{блок_кода1} [ElseIf (*условие2*)]
{блок_кода2}] [Else
{блок_кода3}]

При выполнении инструкции If проверяется истинность условного выражения *условие1*.

Если *условие1* имеет значение $True, то выполняется блок_кода1, после чего выполнение инструкции if завершается. Если *условие1* имеет значение $False, проверяется истинность условного выражения *условие2*. Если *условие2* имеет значение $True, то выполняется блок_кода2 и выполнение инструкции if завершается. Если и *условие1,* и *условие2* имеют значение $False, то выполняется блок_кода3 и

выполнение инструкции if завершается.

Пример 2. использования инструкции if в интерактивном режиме работы. Сначала переменной $a присвоим значение 10:

PS C:\> $a=10

Затем сравним значение переменной с числом 15: PS C:\> If ($a –eq 15) {

>> ‘Значение $a равно 15’

>> }

>> Else {‘ Значение $a не равно 15‘}

>>

Значение $a не равно 15

Из приведенного примера видно также, что в оболочке PS в интерактивном режиме можно выполнять инструкции, состоящие из нескольких строк, что полезно при отладке сценариев.

9.2 Циклы While и Do … While

Самый простой из циклов PS – цикл While, в котором команды выполняются до тех пор, пока проверяемое условие имеет значение $True. Инструкция While имеет следующий синтаксис:

While (условие) {блок_команд}

Цикл Do … While похож на цикл While, однако условие в нем проверяется не до блока команд, а после: Do {блок_команд} While (условие). Например:

PS C:\> $val=0

PS C:\>Do {$val++; $val} While ($val –ne 3) 1

2

3

9.3 Цикл For

Обычно цикл For применяется для прохождения по массиву и выполнения определенных действий с каждым из его элементов. Синтаксис инструкции For:

For (инициация; условие; повторение) {блок_команд}. Пример PS C:\> For ($i=0; $i –lt 3; $i++) {$i }

0

1

2

9.4 Цикл ForEach

Инструкция ForEach позволяет последовательно перебирать элементы коллекций. Самый простой тип коллекции – массив. Особенность цикла ForEach состоит в том, что его синтаксис и выполнение зависят от того, где расположена инструкция ForEach: вне конвейера команд или внутри конвейера.

Инструкция ForEach вне конвейера команд:

В этом случае синтаксис цикла ForEach имеет вид:

ForEach ($элемент in $коллекция) {блок_команд}

При выполнении цикла ForEach автоматически создается переменная

$элемент. Перед каждой итерацией в цикле этой переменной присваивается значение очередного элемента в коллекции. В разделе блок_команд содержатся команды, выполняемые на каждом элементе коллекции. Приведенный ниже цикл ForEach отображает значения элементов массива $lettArr:

```
PS C:\> $lettArr = “a”, “b”,”c”
PS C:\> ForEach ($lett in $lettArr) {Write-Host $lett} a
b c
```

Инструкция ForEach может также использоваться совместно с командлетами, возвращающими коллекции элементов. Например:

```
PS C:\> $ln = 0; ForEach ($f in Dir *.txt) {$ln +=$f.length}
```

В примере создается и обнуляется переменная $ln, затем в цикле ForEach с помощью командлета dir формируется коллекция файлов с расширением txt, находящихся в текущем каталоге. Инструкция ForEach перебирает все элементы этой коллекции, на каждом шаге к текущему файлу выполняется обращение с помощью переменной $f. В блоке команд цикла ForEach к текущему значению переменной $ln добавляется значение свойства Length (размер файла) переменной $f. В результате выполнения цикла в переменной $ln будет получен суммарный размер файлов в текущем каталоге, которые имеют расширение txt.

Инструкция ForEach внутри конвейера команд:

Если инструкция ForEach появляется внутри конвейера команд, то PS использует псевдоним ForEach, соответствующий командлету ForEach- Object. В этом случае фактически выполняется командлет ForEach-Object и не требуется часть инструкции ($элемент in $коллекция), так как элементы коллекции блоку команд предоставляет предыдущая команда конвейера.

Синтаксис инструкции ForEach внутри конвейера команд имеет вид: команда | ForEach {блок_команд}

Рассмотренный выше пример подсчета суммарного размера файлов из текущего каталога для данного варианта инструкции ForEach примет следующий вид:

```
PS C:\> $ln = 0; dir *.txt | ForEach { $ln +=$_.Length}
```

В приведенном примере специальная переменная $_ используется для обращения к текущему объекту конвейера и извлечения его свойств.

10 Управление выводом команд в PowerShell

Рассмотрим, каким образом система формирует строки текста, которые выводятся на экран в результате выполнения той или иной команды (напомним, что командлеты PowerShell возвращают .NET- объекты, которые, как правило, не знают, каким образом отображать себя на экране).

В PowerShell имеется база данных (набор XML-файлов), содержащая модули форматирования по умолчанию для различных типов .NET- объектов. Эти модули определяют, какие свойства объекта отображаются при выводе и в каком формате: списка или таблицы. Когда объект достигает конца конвейера, PowerShell определяет

его тип и ищет его в списке объектов, для которых определено правило форматирования. Если данный тип в списке обнаружен, то к объекту применяется соответствующий модуль форматирования; если нет, то PowerShell просто отображает свойства этого .NET-объекта.

Также в PowerShell можно явно задавать правила форматирования данных, выводимых командлетами, и подобно командному интерпретатору Cmd.exe перенаправлять эти данные в файл, на принтер или в пустое устройство.

10.1 Форматирование выводимой информации

В традиционных оболочках команды и утилиты сами форматируют выводимые данные. Некоторые команды (например, `dir` в интерпретаторе Cmd.exe) позволяют настраивать формат вывода с помощью специальных параметров.

В оболочке PowerShell вывод форматируют только четыре специальных командлета Format (табл. 3). Это упрощает изучение, так как не нужно запоминать средства и параметры форматирования для других команд (остальные командлеты вывод не форматируют).

Таблица 3 - Командлеты PowerShell для форматирования вывода

Командлет	**Описание**
`Format- Table`	Форматирует вывод команды в виде таблицы, столбцы которой содержат свойства объекта (также могут быть добавлены вычисляемые столбцы). Поддерживается возможность группировки выводимых данных
`Format- List`	Вывод форматируется как список свойств, в котором каждое свойство отображается на новой строке. Поддерживается возможность группировки выводимых данных
`Format- Custom`	Для форматирования вывода используется пользовательское представление (view)
`Format- Wide`	Форматирует объекты в виде широкой таблицы, в которой отображается только одно свойство каждого объекта

Как уже отмечалось, если ни один из командлетов Format явно не указан, то используется модуль форматирования по умолчанию, который определяется по типу отображаемых данных. Например, при выполнении командлета Get-Service данные по умолчанию выводятся как таблица с тремя столбцами (Status, Name и DisplayName): (рис.11) PS C:\> Get-Service

Status	Name	DisplayName
Stopped	Alerter	Оповещатель
Running	ALG	Служба шлюза уровня приложения
Stopped	AppMgmt	Управление приложениями
Stopped	aspnet_state	ASP.NET State Service
Running	Ati HotKey	Poller Ati HotKey Poller
Running	AudioSrv	Windows Audio
Running	BITS	Фоновая интеллектуальная служба пер...
Running	Browser	Обозреватель компьютеров
Stopped	cisvc	Служба индексирования
Stopped	ClipSrv	Сервер папки обмена
Stopped	clr_optimizatio...	.NET Runtime Optimization Service v...
Stopped	COMSysApp	Системное приложение COM+
Running	CryptSvc	Службы криптографии
Running	DcomLaunch	Запуск серверных процессов DCOM
Running	Dhcp	DHCP-клиент

Рисунок 11- Таблица с тремя столбцами (Status, Name и DisplayName)

Для изменения формата выводимых данных нужно направить их по конвейеру соответствующему командлету `Format`. Например, следующая команда выведет список служб с помощью командлета `Format-List`:

PS C:\> Get-Service | Format-List (рис.12.)

```
PS C:\> Get-Service | Format-List

Name                  : Alerter
DisplayName           : Оповещатель
Status                : Stopped
DependentServices     : {}
ServicesDependedOn    : {LanmanWorkstation}
CanPauseAndContinue   : False
CanShutdown           : False
CanStop               : False
ServiceType           : Win32ShareProcess

Name                  : ALG
DisplayName           : Служба шлюза уровня приложения
Status                : Running
DependentServices     : {}
ServicesDependedOn    : {}
CanPauseAndContinue   : False
CanShutdown           : False
CanStop               : True
ServiceType           : Win32OwnProcess
. . .
```

Рисунок 12- Список служб с помощью командлета `Format-List`

При использовании формата списка выводится больше сведений о каждой службе, чем в формате таблицы (вместо трех столбцов данных о каждой службе в формате списка выводятся девять строк данных). Однако это вовсе не означает, что

командлет Format-List извлекает дополнительные сведения о службах. Эти данные содержатся в объектах, возвращаемых командлетом Get-Service, однако командлет Format- Table, используемый по умолчанию, отбрасывает их, потому что не может вывести на экран более трех столбцов.

При форматировании вывода с помощью командлетов Format- List и Format-Table можно указывать имена свойства объекта, которые должны быть отображены (напомним, что просмотреть список свойств, имеющихся у объекта, позволяет рассмотренный ранее командлет Get-Member). Например:

```
PS C:\> Get-Service | Format-List Name, Status, CanStop

Name : Alerter Status : Stopped CanStop : False

Name: ALG Status : Running CanStop : True

Name: AppMgmt Status : Stopped CanStop : False
. . .
```

Вывести все имеющиеся у объектов свойства можно с помощью параметра *, например:

```
PS C:\> Get-Service | Format-table *
```

10.2 Перенаправление выводимой информации

В оболочке PowerShell имеются несколько командлетов, с помощью которых можно управлять выводом данных. Эти командлеты начинаются со слова Out, их список можно получить с помощью команлета:

```
PS C:\> Get-Command out-* | Format-Table Name

Name
----
Out-Default Out-File Out-Host
Out-Null Out-Printer Out-String
```

По умолчанию выводимая информация передается командлету Out-Default, который, в свою очередь, делегирует всю работу по выводу строк на экран командлету Out-Host. Для понимания данного механизма нужно учитывать, что архитектура PowerShell подразумевает различие между собственно ядром оболочки (интерпретатором команд) и главным приложением (host), которое использует это ядро. В принципе, в качестве главного может выступать любое приложение, в котором реализован ряд специальных интерфейсов, позволяющих корректно интерпретировать получаемую от PowerShell информацию. В нашем случае главным приложением является консольное окно, в котором мы работаем с оболочкой, и командлет Out-Host передает выводимую информацию в это консольное окно.

Параметр Paging командлета Out-Host, подобно команде more интерпретатора Cmd.exe, позволяет организовать постраничный вывод информации, например:

Get-Help Get-Process –Full | Out-Host –Paging

10.3 Сохранение данных в файл

Командлет `Out-File` позволяет направить выводимые данные вместо окна консоли в текстовый файл. Аналогичную задачу решает оператор перенаправления (>), однако командлет `Out-File` имеет несколько дополнительных параметров, с помощью которых можно более гибко управлять выводом: задавать тип кодировки файла (параметр `Encoding`), задавать длину выводимых строк в знаках (параметр `Width`), выбирать режим перезаписи файла (параметр `Append`). Например, следующая команда направит информацию о выполняющихся на компьютере процессах в файл C:\Process.txt, причем данный файл будет записан в формате ASCII:

Get-Process | Out-File -FilePath C:\Process.txt -Encoding ASCII

10.4 Подавление вывода

Командлет `Out-Null` служит для поглощения любых своих входных данных. Это может пригодиться для подавления вывода на экран ненужных сведений, полученных в качестве побочного эффекта выполнения какой-либо команды. Например, при создании каталога командой `mkdir` на экран выводится его содержимое: (рис.13)

PS C:\> mkdir spo

Каталог: Microsoft.PowerShell.Core\FileSystem::C:\

Mode	LastWriteTime	Length	Name
d----	03.01.2015 1:01		spo

Рисунок 13-Создании каталога командой `mkdir` на экран выводится его содержимое

Если эта информация не нужна, то результат выполнения команды `mkdir` необходимо передать по конвейеру командлету `Out-Null`: mkdir spo | Out-Null

10.5 Преобразование данных в формат html, сохранение в файле и просмотр результатов

Для преобразования данных в формат html служит командлет `Convertto-html`. `Параметр Property` определяет свойства объектов, включаемые в выходной документ. Например, для получения списка выполняемых процессов в формате html, включающего имя процесса и затраченное время CPU и записи результата в файл processes.html можно использовать команду

Get-Process | Convertto-html -Property Name, CPU > Processes.htm Для просмотра содержимого файла можно использовать командлет Invoke-Item “имя документа”

Например Invoke-Item “processes.htm”

10.6 Инвентаризация и диагностика Windows-компьютеров

Для вывода сведений о процессоре ПК служит командлет Get- wmiobject

Get-wmiobject -Class Win32_Processor | Format-list *

10.7 Командлеты для измерения свойств объектов

Для иэмерения времени выполнения командлетов PS служит командлет Measure-Command

В качестве примера рассмотрим получение времени выполнение командлета dir

(Measure-Command {dir}).TotalSeconds

Для получения статистических данных служит командлет Measure- Object. Для числовых массивов с его помощью можно получить максимальное, минимальное, среднее значение элементов массива и их сумму. Если имеется инициализированный массив ms, для указанной цели используется командлет

$ms | measure-object –maximum –minimum –average -sum

МЕТОДИКА ВЫПОЛНЕНИЯ

1. Ознакомиться с теоретическими сведениями.
2. Вывести содержимое каталога Windows (для вариантов 5 и 10 – и подкаталогов) по указанному в табл. 4 формату на экран и в текстовый файл.

Таблица 4 - Варианты заданий для студентов по списку

№ Студента	Что выводить (имена, размер, дата создания,атрибуты)	Сортировать по	Условие отбора
1, 6	Только файлы	По размеру	Размер>10000
2, 7	Файлы и подкаталоги	По дате	Первые буквы имени SY
3, 8	Только подкаталоги	Именам	Последняя буква имени S или T
4, 9	Только файлы bmp	По размеру	Размер >50000
5, 10	Только файлы jpg	Именам	Любые

Рекомендуется использовать фильтр по Extension или Attributes (в зависимости от варианта задания)

3. Вывести в текстовый файл список свойств процесса, возвращаемый командлетом Get-process и на экран – их общее количество.
4. Создать текстовый файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в табл. 5 параметра. Имена параметров процессов указаны так же в табл. 5.

Таблица 5 - Варианты заданий для для студентов по списку

№ Студента	Список выводимых параметров процессов	Сортировать по значению параметра	Вывести процессы, у которых
1, 4	Имя процесса, BasePriority, Company	Имя процесса	BasePriority > 7
2, 6	Id, Имя процесса, время старта, Handles	Время старта	Id > 40
3, 5	Имя процесса, Id, PriorityClass, UserprocessorTime, TotalProcessorTime	TotalProcessorTime	Id > 100
7, 8	Имя процесса, PriorityClass, ProductVersion, Id	Имя процесса	Id > 100
9, 10	Id, Имя процесса, WorkingSet, CPU	Id	CPU > 5

5. Создать HTML-файл, содержащий список выполняемых процессов, упорядоченный по возрастанию указанного в табл.5 параметра. Имена параметров процессов указаны в табл. 5.

6. Найти суммарный объем всех графических файлов (bmp, jpg), находящихся в каталоге Windows и всех его подкаталогах.

7. Вывести на экран сведения о ЦП компьютера.

8. Найти максимальное, минимальное и среднее значение времени выполнение командлетов dir и ps

9. Выполнить индивидуальные задания для студентов согласно (табл. 6).

Таблица 6 - Варианты заданий для студентов по списку

№№	Содержание задания – разработать командлет для:
1	-вычисления факториала от целочисленной переменной с именем numb -нахождения минимального и максимального значений чисел, хранящихся в файле nn.txt
2	-нахождения количества различных чисел, хранящихся в файле nn.txt 2. -нахождения количества наибольших чисел, хранящихся в файле nn.txt
3	-нахождения количества положительных чисел, хранящихся в файле nn.txt -нахождения количества четных чисел, хранящихся в файле nn.txt
4	-нахождения в заданном каталоге файла наибольшего размера -нахождения в заданном каталоге трех файлов наименьшего размера
5	-нахождения среди выполняющихся процессов имен процессов, выполняющихся в двух или более экземплярах -нахождения среди выполняющихся процессов имени процесса, запущенного последним

6	-нахождения среди выполняющихся процессов имен трех процессов, использовавших более всего процессорного времени -нахождения среди выполняющихся процессов имени процесса с наибольшим размером рабочего множества страниц
7	-нахождения среди выполняющихся процессов имен процессов с наименьшим значением BasePriority -нахождения среди выполняющихся процессов имен процессов, у которых значения параметра WorkingSet одинаковы
8	-проверки наличия в текущем каталоге файлов одинакового размера. Если такие файлы есть – вывести их имена -нахождения среди выполняющихся процессов имен процессов с наибольшим значением приоритета
9	-нахождения в каталоге windows\system32 имен трех dll наибольшего и наименьшего размеров -нахождения в каталоге windows\system32 имен трех dll с самой ранней датой создания
10	-нахождения среди выполняющихся процессов имен трех процессов, работающих в системе дольше всего -нахождения среди выполняющихся процессов имен процессов, имеющих одинаковые ProductVersion
11	-разбиения текстового файла, содержащего четное количество строк, на два текстовых файла, в каждый из которых записать одинаковое количество строк -нахождения в каталоге windows и его подкаталогах имен библиотек dll из шести символов, начинающихся на mfc и заканчивающихся буквой u
12	-нахождения в каталоге windows и его подкаталогах имен файлов, записанных русскими буквами и имеющих расширение jpg. -нахождения в текстовом файле, содержащем слова английского и русского языков (одно слово в строке), слов русского языка и вывода их на экран
13	-нахождения в текстовом файле, содержащем слова английского и русского языков и числа (одно слово или число в строке), чисел и вывода их на экран -нахождения в текстовом файле, содержащем слова английского и русского языков (одно слово в строке), слов, написанных с заглавной буквы и вывода их на экран

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Типы команд PowerShell (PS).
2. Имена и структура командлетов.
3. Псевдонимы команд.
4. Просмотр структуры объектов.

5. Фильтрация объектов в конвейере. Блок сценария.
6. Какую информацию выводит команда Get-Help * ?
7. Командлеты для форматирования выводимой информации.
8. Перенаправление выводимой информации.
9. Управляющие инструкции PS.
10. Назначение регулярных выражений.
11. Сохранение данных в текстовом файле и html-файле.
12. Получение справочной информации в PS.
13. Как создать массив в PS?
14. Как объединить два массива?
15. Как увеличить размер созданного в PS массива?
16. Как ввести данные в массив?
17. Использование командлета Out-Null.
18. Оператор PowerShell –match.
19. Использование символа ^ в командлетах.
20. Использование символа $ в командлетах.
21. Количественные модификаторы (квантификаторы).
22. Использование групп захвата.
23. Командлеты для измерения свойств объектов.

ЛАБОРАТОРНАЯ РАБОТА №3 (4 часа)
АДМИНИСТИРОВАНИЕ POWERSHELL ОС WINDOWS
Контрольная работа

Цель работы: практическое применение скриптов PowerShell

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

По умолчанию PowerShell установлвлен во всех версиях Windows, начиная с Windows 7 SP1 и Windows Server 2008 R2 SP1. В следующей таблице 1 представлен список актуальных версий PowerShell.

Вы можете вручную установить более новую версию PowerShell в предыдущих версиях Windows. Для этого нужно скачать и установить соответствующую версию **Windows Management Framework** (PowerShell входит в его состав).

Стоит обратить внимание, что последние 2 года Microsoft приостановила развитие классического Windows PowerShell (выпускаются только исправления ошибок и безопасности) и сфокусировалась на открытом кроссплатформенном PowerShell Core.

Отличия Windows PowerShell от PowerShell Core:

- Windows PowerShell основан на NET Framework (например, для PowerShell 5 требуется .NET Framework v4.5, нужно убедиться, что он установлен). PowerShell Core основан на .Net Core;
- Windows PowerShell работает только на ОС семейства Windows, а PowerShell Core является кроссплатформенным и будет работать в Linux;
- В PowerShell Core нет полной совместимости с Windows PowerShell, однако Microsoft работает над улучшением обратной совместимости со старыми командлетами и скриптами (перед переходом на PowerShell Core рекомендуется протестировать работу старых PS скриптов). В PowerShell 7 обеспечивается максимальная совместимсть с Windows PowerShell.
- Редактор PowerShell ISE нельзя использовать для отладки скриптов PowerShell Core (но можно использовать Visual Studio Code)
- т.к. Windows PowerShell более не развивается, рекомендуется постепенно мигрировать на PowerShell Core.

1 Запуск скриптов

По-умолчанию настройки Windows запрещают запуск скриптов PowerShell. Это необходимо для предотвращения запуска вредоносного кода на PowerShell. Настройки политик запуска PowerShell скриптов определяются в Execution Policy. В этой статье мы рассмотрим доступные политики запуска PS скриптов, как изменить Execution Policy и настроить политики использования PowerShell скриптов на компьютерах в домене. Доступны следующие значения PowerShell Execution Policy:

- **Restricted** – запрещен запуск скриптов PowerShell, можно выполнять только интерактивные команды в консоли;

- **AllSigned** – разрешено выполнять только подписанные PS скрипты с цифровой подписью от доверенного издателя (можно подписать скрипт самоподписанным сертификатом и добавить его в доверенные). При запуске недоверенных скриптов появляется предупреждение: «Do you want to run software from this untrusted publisher? File .ps1 is published by CN=test1 and is not trusted on your system. Only run scripts from trusted publishers».
- **RemoteSigned** – можно запускать локальные PowerShell скрипты без ограничения. Можно запускать удаленные PS файлы с цифровой подписью (нельзя запустить PS1 файлы, скачанные из Интернета, запущенные из сетевой папки по UNC пути и т.д.);
- **Unrestricted** – разрешен запуск всех PowerShell скриптов;
- **Bypass** – разрешён запуск любых PS файлов (предупреждения не выводятся) – эта политика обычно используется для автоматического запуска PS скриптов без вывода каких-либо уведомлений (например, при запуске через GPO, SCCM, планировщик и т.д.) и не рекомендуется для постоянного использования;
- **Default** – сброс настроек выполнения скриптов на стандартную. В Windows 10 значение политики выполнения PowerShell по-умолчанию Restricted, а в Windows Server 2016 — RemoteSigned.
- **Undefined** – не задано. Применяется политика Restricted для десктопных ОС и RemoteSigned для серверных.

1.1 Запуск скриптов PowerShell с помощью Execution Policy

Чтобы изменить текущее значение политики запуска PowerShell скриптов, используется командлет **Set-ExecutionPolicy**.

Области действия политик выполнения скриптов PowerShell (scopes):

- **MachinePolicy** – действует для всех пользователей компьютера, настраивается через GPO;
- **UserPolicy** – действует на пользователей компьютера, также настраивается через GPO;
- **Process** — настройки ExecutionPolicy действует только для текущего сеанса PowerShell.exe (сбрасываются при закрытии процесса);
- **CurrentUser** – политика ExecutionPolicy применяется только к текущему пользователю (параметр из ветки реестра HKEY_CURRENT_USER);
- **LocalMachine** – политика для всех пользователей компьютера (параметр из ветки реестра HKEY_LOCAL_MACHINE).

Значение политики выполнения, которые задаются с помощью командлета Set-ExecutionPolicy для областей CurrentUser и LocalMachine, хранятся в реестре.

Например, командлет:

Set-ExecutionPolicy -Scope LocalMachine -ExecutionPolicy Restricted –Force

В ветке реестра HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell можно проверить значение REG_SZ параметра **ExecutionPolicy**, рисунок 1. Оно изменилось на Restricted (допустимые значения параметра Restricted, AllSigned, RemoteSigned, Bypass, Unrestricted и Undefined).

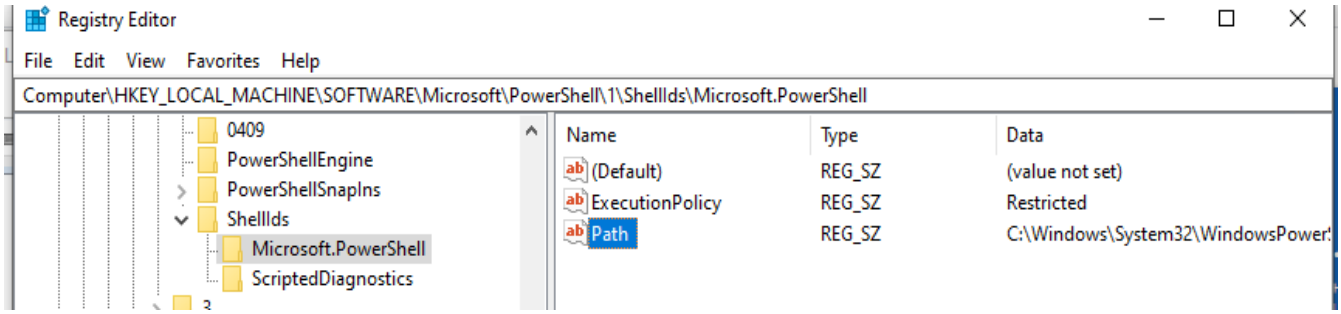

Рисунок 1 - Значение политики выполнения в реестре

Аналогичные настройки для области CurrentUser находятся в разделе реестра пользователя HKEY_CURRENT_USER\SOFTWARE\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell.

То есть вы можете распространить нужные настройки политики исполнения скриптов через реестр с помощью Group Policy Preferences.

Чаще всего в корпоративной среде используется ExecutionPolicy со значением **AllSigned** на уровне LocalMachine. Это обеспечивает максимальный баланс между безопасностью и удобством. Для личного пользования на компьютере можно использовать RemoteSigned. Ну а Bypass политику лучше использовать только для запуска отдельных задач (например, для запуска скриптов через GPO или заданий планировщика).

2 Настройка PowerShell Execution Policy с помощью групповых политик

Одним из основных инструментов тонкой настройки параметров пользователя и среды Windows являются **групповые политики — GPO (Group Policy Object)**. На сам компьютер и его пользователей могут действовать доменные групповые политики (если компьютер состоит в домене Active Directory) и локальные (эти политики настраиваются локально на компьютере). Однако некорректная настройка некоторых параметров GPO может привести к различным проблемам: невозможность подключить принтер, запрет на подключение USB накопителей, ограничение сетевого доступа некорректными настройками брандмауэра Windows, запрета на установку или запуск любых приложений (через политики SPR или AppLocker) или на локальный или удаленный вход на компьютеры.

Можно настроить политику выполнения PowerShel скриптов на серверах или компьютерах домена с помощью групповых политик. Очень часто для тонкой настройки параметров Windows используется консоль редактора групповых политик (**gpedit.msc**). Однако в домашних редакциях Windows 10 Home консоль редактор политик отсутствует(в отличии от Windows 10 Pro и Enterprise).

Редактор групповой политики это отдельная оснастка, которая по сути представляет собой графическую надстройку для удобного управления параметрами Windows в реестре. При изменении настроек какой-то политики, редактор сразу вносит изменения в связанный параметр реестра. Вместо того, чтобы искать

необходимый ключ и вручную править параметр реестра, гораздо проще найти и отредактировать настройку в редакторе gpedit.msc. В редакторе GPO содержится более двух тысяч настроек Windows, которые расположены в стройной иерархии, имеют подробное описание и предлагают для выбора предопределенные опции настройки.

Соответствие между политиками и ключами реестра можно найти в XLSX документе Microsoft Group Policy Settings Reference Spreadsheet Windows. Версию документа можно скачать на официальном сайте. В этом документе в таблице вы можете найти какой ключ реестра правиться той или иной настройкой конкретной политики.

Если администратор не может локально войти в систему, или не знает точно какая из примененных им настроек GPO вызывает проблему, приходится прибегать к аварийному сценарию сброса настроек групповых политик на настройки по-умолчанию. В “чистом” состоянии ни один из параметров групповых политик не задан.

3 Сертификаты и PowerShell

Наличие цифровой подписи у скрипта или исполняемого файла позволяет пользователю удостовериться, что файл является оригинальным, и в том, что его код не был изменен третьими лицами. В PowerShell также имеется встроенная возможность подписать файл скрипта *.ps1.

Подписать скрипт PowerShell можно с помощью специального типа сертификата **Code Signing**. Этот сертификат может быть получен от внешнего коммерческого центра сертификации, внутреннего корпоративного CA или самоподписанного сертификата.

4 Работа планировщика

Планировщик Windows служит для запуска программ по расписанию или в ответ на какие-то события. Рапускают различные скрипты, например PowerShell.

Планировщик Windows находится в панели управления в разделе “Система и безопасность”-”Администрирование”-”Расписание выполнения задач”.

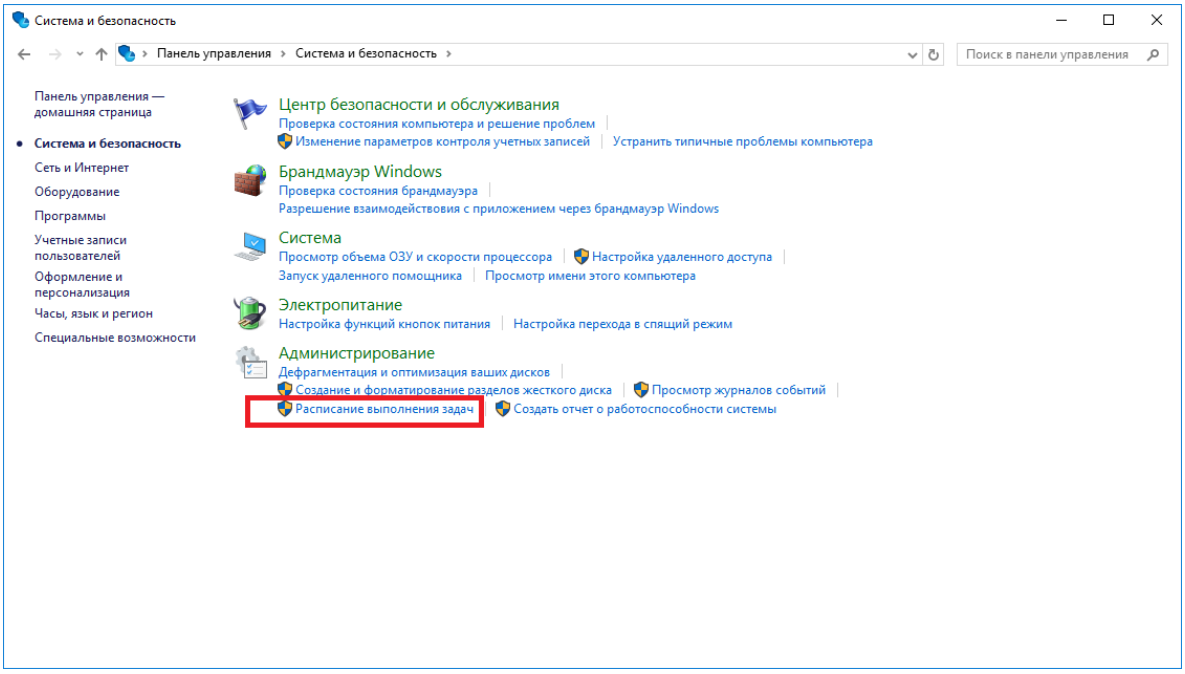

Рисунок 2 – Планировщик Windows

Первое, на что следует обратить внимание еще до создания задач - запущена ли служба «Планировщик заданий». Эта служба могла быть отключена при настройке системных служб для освобождения памяти в то время, когда использование Планировщика задач не было нужно. Теперь, если запланированные задания будут выполняться регулярно, то этот сервис надо переключить в автоматический режим запуска.

Чтобы убедиться, что сервис запущен, зажмите поочередно клавиши Win + R, и в открывшемся окошке "Выполнить", нужно набрать команду services.msc и нажать кнопку "ОК". Откроется окно, показанное на рисунке 3.

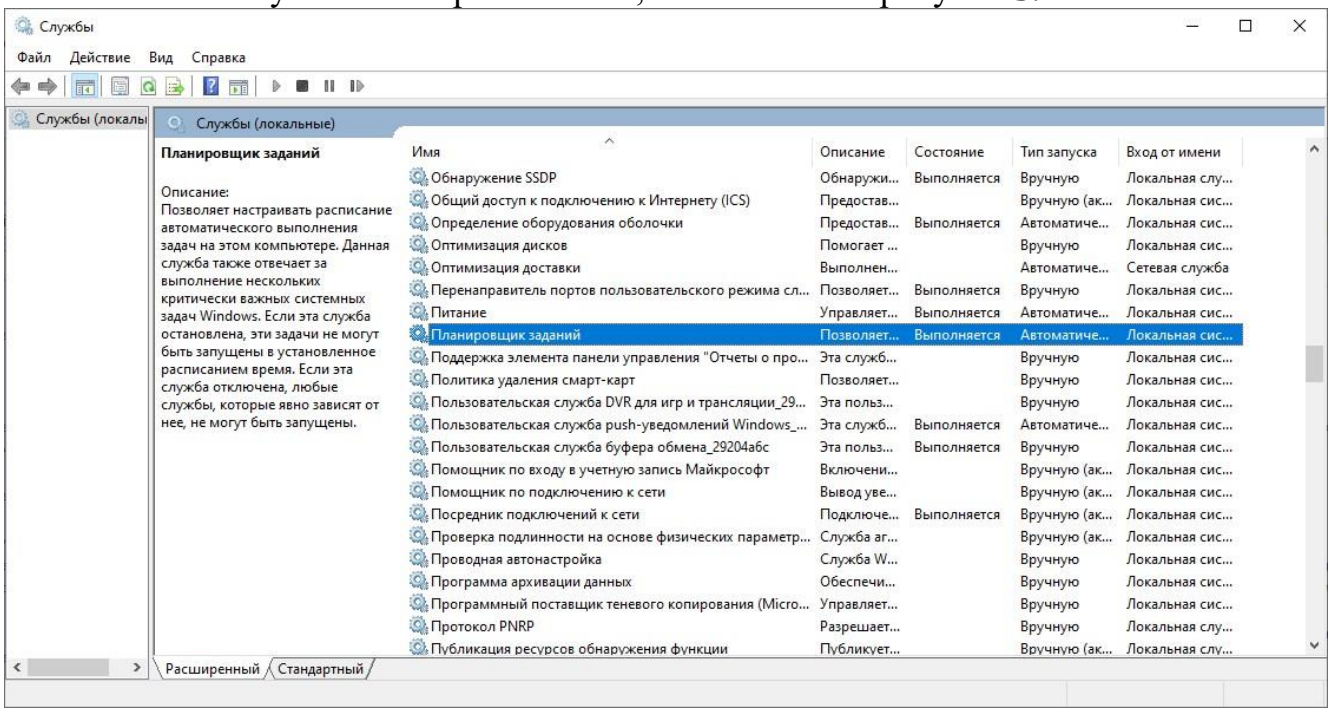

Рисунок 3 – Службы Windows

Напротив, службы «Планировщик заданий» в столбце «Состояние» должно быть значение «Выполняется» («Работает» в старых версиях Windows), а в столбце "Тип запуска" - "Автоматически". Если это не так, то дважды щелкните по имени службы и в открывшемся окне скорректируйте значения на те, которые указаны выше (для этого нужно иметь привилегии администратора, т. е. ваша учетная запись должна быть из группы Администраторы).

После того, как служба запущена и тип ее запуска скорректирован на автоматический, служба будет стартовать при загрузке системы и задания буду выполняться в соответствии с расписанием.

Для создания задания на запуск скрипта необходимо иметь файл скрипта для запуска, «Создать задачу» в меню «Действия» в правой части экрана.

На первой надо ввести имя задачи и, возможно, развернутое описание. Далее - от какого пользователя она будет выполняться, должна ли выполняться только после входа в систему (выполнять для пользователей, вошедших в систему) или вообще для всех пользователей. Обратите внимание, что если запускаемые программы используют сетевые диски и папки, то необходимо указать пароль к учетной записи, от которой будет запускаться задача. Выполнить с наивысшими правами означает «Запускать от имени администратора». Также можно сделать чтобы задача вообще не отображалась при запуске - так называемая скрытая задача.

На закладке «Триггеры» можно задать одно или несколько расписаний запуска. Обратите внимание, что в меню «Начать задачу» есть различные типы запуска задачи: по расписанию, при простое, при входе в систему и т.д.

Также можно указать периодичность запуска задачи, отложить запуск, задать сроки действия задачи, и т.д.

На закладке «Действия» указываем какая программа будет запускаться и при необходимости ее параметры. Также можно просто вывести сообщение на экран или отправить сообщение по электронной почте (меню действие).

На вкладке «Условия» можно задать дополнительные условия работы задачи - например «Запускать только при питании от сети» - актуально для устройств, которым нужна экономия питания - например ноутбуки – «разбудить компьютер для запуска задания» и т.д.

На последней вкладке можно задать дополнительные параметры, например перезапуск задачи при сбое или запуск задачи после пропуска (компьютер был выключен и задание не смогла исполниться в назначенное время).

После нажатия кнопки «Готово» задание создастся и помещается в библиотеку планировщика заданий. Здесь можно им управлять (запускать, останавливать, редактировать, включать/выключать и т.д.).

Также у планировщика есть и другие функции - например импорт/экспорт заданий, ведение журналов запуска, создание простый заданий (задаются только имя, описание, программа, пользователь и расписание) и так далее.

МЕТОДИКА ВЫПОЛНЕНИЯ

1. Определить какая версия PowerShell у вас установлена с помощью консоли.
2. Определить установленную версию PowerShell через реестр.
3. Получить текущее значение политики выполнения скриптов PowerShell на

компьютере.

4. Изменить текущее значение политики запуска PowerShell скриптов – разрешить запуск локальных скриптов, определить область применения политики (выбрать любую), проверить текущие настройки ExecutionPolicy для всех областей. Настроить политику исполнения скриптов PowerShell, разрешив запуск только подписанных скриптов.

5. Проверить хранилища сертификатов Windows на наличие недоверенных корневых сертификатов.

6. Выполнить все задания (преподаватель по своему выбору опрашивает 3 задания):

1) Организовать запуск PowerShell скрипта при загрузке компьютера с помощью групповой политики.

2) С помощью PowerShell создать новые задания планировщика Windows, экспортировать задания в xml-файл. Задача - создать задание планировщика которое бы запускалось в определенное время, задание должно выполнять некий PowerShell скрипт или команду.

Примечание. PowerShell предоставляет возможность экспортировать текущие настройки любого задания планировщика в текстовый XML файл. Таким образом можно выгрузить параметры любого задания и распространить задание любой сложности на другие компьютеры сети. Экспорт задания может быть выполнен как из графического интерфейса Task Scheduller, так и из командой строки PowerShell.

3) Проверка свободного места на диске с помощью PowerShell скрипта. Написать скрипт PowerShell для мониторинга свободного места на диске компьютера с Windows и оповещения администратора (всплывающим сообщением или письмом), если превышено пороговое значение.

4) Добавить в контекстное меню проводника File Explorer для файлов с расширением *.ps1, пункт, позволявший запустить скрипт PowerShell с правами администратора.

Примечание. В Windows скрипты PowerShell (расширение .**PS1**) по умолчанию не ассоциированы с исполнимым файлом PowerShell.exe. При двойном щелке по файлу сценария PS1 открывается окно тестового редактора notepad.exe. Запустить файл PS1 на выполнение в среде PowerShell можно из контекстного меню проводника, выбрав пункт **Run With PowerShell**. Однако такой сценарий запускается в рамках сессии пользователя, без прав администратора. Хотя для тех же файлов скриптов .bat, .cmd, имеется отдельный пункт меню Run As administrator. В случае с PowerShell приходится открывать консоль Power Shell с повышенными правами и указывать полный путь к файлу скрипта, что не удобно.

5) Создать PowerShell код, который можно использовать, чтобы проверить, запущён ли текущий скрипт в режиме “Run as Administrator” или нет. Если для выполнения некоего скрипта PowerShell нужно, чтобы он был запушен с правами администратора, необходимо прямо в PS коде выполнить проверку на наличие административных привилегий у текущего процесса. А так же прямо из скрипта PowerShell запросить повышение привилегий.

6) Сделать резервную копию, бэкап (экспорт) драйверов и восстановить их с помощью PowerShell.

Примечание. После переустановки или чистой установки Windows пользователь сталкивается с необходимостью установки актуальных версий драйверов для устройств, установленных в компьютере. Сразу после переустановки системы пользователю приходится вручную качать драйвера с сайта производителя или использовать различные драйвер-паки (в комплекте с которыми как правило идет разный мусор и рекламное ПО). Перед переустановкой Windows вы можете создать резервную копию всех драйверов, установленных в системе. В дальнейшем с помощью резервной копии вы сможете быстро установить все необходимые драйвера уже в чистой системе.

7) С помощью PowerShell получить для просмотра и изменения настроек BIOS на ПК, например, включить/отключить загрузку компьютера с USB устройств.

8) Вывести список дисков и разделов на них с помощью PowerShell:

-выведите список дисков, доступных на логическом уровне в системе,

-выведите информацию о физических дисках (характеристики и состояние физических дисков на ПК), определите тип подключенного накопителя SSD или HDD,

-вывести список разделов на всех дисках,

-вывести список всех логических разделов в Windows,

- найти диск в офлайн состоянии и инициализировать его,

- создать новый раздел на диске, переименовать его, отформатировать его в файловой системе NTFS и задать метку тома DBData,

- удалить раздел диска.

Примечание. Нумерация дисков начинается с 0, а нумерация разделов с 1.

9) Необходимо из скрипта PowerShell оповестить пользователя об определенном событии или необходимости выполнить определенное действие. Например, вывести уведомление о завершении какого-либо длительного PoSh скрипта, или об наступлении какого-то важного события.

10) С помощью PowerShell создать Zip-архив и распаковать его.

11) Управление разрешениями на объекты файловой системы NTFS из PowerShell:

- вывести текущего владельца папки (файла) и список назначенных NTFS разрешений,

- предоставить конкретному пользователю и группе полные права на папку,

- предоставить права только на верхнем уровне и не изменять разрешения на вложенные объекты (только на папку)

- убрать NTFS доступ к папке для пользователя или группы,

- лишить указанную учетную запись прав на все вложенные объекты в указанной папке,

- назначить учетную запись Administrator владельцем всех вложенных объектов в каталоге,

- проверить эффективные NTFS разрешения на конкретный файл.

Примечание. Для управления доступом к файлам и папкам в Windows на каждый объект файловой системы NTFS (каталог или файл) назначается специальный ACL (Access Control List, список контроля доступа). В ACL объекта задаются доступные операции (разрешения), которые может совершать с этим объектом пользователь и/или группы.

12) Управление службами Windows с помощью PowerShell:

- получить список служб и их состояние,
- вывести имя, статус и доступные возможности любой службы,
- вывести службы, необходимые для запуска текущей службы,
- вывести зависимые службы от текущей,
- проверить, что в системе имеется указанная служба,
- остановите любую службу, включая зависимые,
- запустите любую службу, включая зависимые,
- отобразите список всех служб, работа которых может быть приостановлена,
- приостановите любую службу и возобновите ее работу,
- измените тип запуска службы – ручной, автоматический,
- создайте новую службу TestService в Windows. Для новой службы требуется указать имя и исполняемый файл,
- получите информацию о режиме запуска и описание службы,
- получите имя учетной записи, которая используется для запуска службы TestService,
- изменените имени и пароль учетной записи указанной службы,
- удалите службу.

Примечание. Существует восемь основных командлетов Service, предназначенных для просмотра состояния и управления службами Windows:

- Get-Service — позволяет получить службы на локальном или удаленном компьютере, как запущенные, так и остановленные;
- New-Service – создать службу. Создает в реестре и базе данных служб новую запись для службы Windows;
- Restart-Service – перезапустить службу. Передает сообщение об перезапуске службы через Windows Service Controller
- Resume-Service – возобновить службы. Отсылает сообщение о возобновлении работы диспетчеру служб Windows;
- Set-Service — изменить параметры локальной или удаленной службы, включая состояние, описание, отображаемое имя и режим запуска. Этот командлет также можно использовать для запуска, остановки или приостановки службы;
- Start-Service – запустить службу;
- Stop-Service – остановить службу (отсылает сообщение об остановке диспетчеру служб Windows);
- Suspend-Service приостановить службу. Приостановленная служба по-прежнему выполняется, однако ее работа прекращается до возобновления работы службы, например с помощью командлета Resume-Service.

13) Отправить письмо с указанной темой и вложением из скрипта PS.

Использовать для письма кодировку UTF8.

14) Создайте самоподписанный сертификат в PowerShell и импортируйте его в персональное хранилище компьютера. Можно использовать командлет **New-SelfSignedCertificate**, входящий в состав модуля **PKI** (Public Key Infrastructure). С помощью командлета Get-ChildItem можно вывести все параметры созданного сертификата по его отпечатку (Thumbprint). Задать срок действия сертификата – 3 года. Ключ сертификата или сам файл сертификата подготовить для распостранения на все ПК.

Примечание. Самоподписанные сертификаты рекомендуется использовать в тестовых целях или для обеспечения сертификатами внутренних интранет служб.

15) Создать самоподписанный сертификат для подписывания кода приложений, проверить доверенность сертификата, настроить и подписать им скрипт PS.

ПРИЛОЖЕНИЕ А
Образец оформления лабораторной работы

ЛАБОРАТОРНАЯ РАБОТА №1
ИНТЕРПРЕТАТОР КОМАНДНОЙ СТРОКИ ОС MS WINDOWS
Внешние и внутренние команды

Выполнил: Иванов И.И., гр. ВКБ-31

Проверил: _____

(ф.и.о, подпись, дата)

Цель работы – знакомство с возможностями интерпретатора командной строки и командами MS Windows

ХОД РАБОТЫ

1. ...
2. … и т.д.

Вывод:…

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. ...
2. … и т.д.

ПРИЛОЖЕНИЕ Б

Образец оформления титульного листа Журнала лабораторных работ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Факультет Информатика и вычислительная техника

Кафедра «Кибербезопасность информационных систем»

ЖУРНАЛ ЛАБОРАТОРНЫХ РАБОТ

по дисциплине_____

Обучающийся _____ _____
подпись, дата

Группа _____

Направление _____

Профиль _____

Проверил: _____кт.н., доцент Язвинская Н.Н.
отметка, подпись, дата

Ростов-на-Дону

202_