Міністерство освіти і науки України
Чорноморський національний університет імені Петра Могили

Г. В. Горбань

МЕТОДИ ПРОГРАМУВАННЯ ПІД ОПЕРАЦІЙНІ СИСТЕМИ

Методичні вказівки

Випуск 365

ЧНУ
ВИДАВНИЦТВО
Миколаїв – 2021

УДК 004.451.9
Г 67

Рекомендовано вченою радою Чорноморського національного університету імені Петра Могили (протокол № 5 від 10 червня 2021 р.).

Рецензент:
Любченко В. В. доктор технічних наук, професор, професор кафедри системного програмного забезпечення Державного університету «Одеська політехніка».

Г 67 **Горбань Г. В.** Методи програмування під операційні системи : методичні вказівки. – Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. – 92 с. (Методична серія ; вип. 365).

Методичні вказівки містять опис 14 лабораторних робіт, в яких розглядаються засоби інструментів автоматизації рутинних задач в операційній системі Windows. Лабораторні роботи 1–4 розглядають командну оболонку cmd, що історично з'явилась в операційній системі першою. Лабораторні роботи 5–8 присвячено серверу сценаріїв Windows Script Host (WSH) та програмуванню скриптів на мові Jscript з його використанням. Лабораторні роботи 9–14 розглядають сучасну командну оболонку Windows PowerShell, що активно використовується у найновіших версіях операційної системи Windows. Методичні вказівки призначено для студентів спеціальності 121 – «Інженерія програмного забезпечення», а також можуть бути корисними для студентів інших спеціальностей галузі знань 12 – «Інформаційні технології».

УДК 004.451.9

ISSN 1811-492X

Зміст

Лабораторна робота № 1. Операційна система MS-DOS. основні поняття і команди 5

Лабораторна робота № 2. Створення і видалення файлів і каталогів у MS-DOS 11

Лабораторна робота № 3. Копіювання і перейменування файлів і каталогів у MS-DOS 14

Лабораторна робота № 4. Програмування командних Bat-файлів, обробка аргументів, мітки, оператори переходу і розгалуження, організація циклів 18

Лабораторна робота № 5. Основи програмування скриптів WSH. основні методи об'єкта WSCRIPT. Запис інформації у документи Word та Excel зі скрипта 22

Лабораторна робота № 6. Використання об'єкту Filesystemobject для роботи з файловою системою. Методи роботи з файлами та каталогами у сценаріях WSH 29

Лабораторна робота № 7. Створення ярликів за допомогою сценаріїв WSH. об'єкт Drive для отримання інформації про диск 37

Лабораторна робота № 8. Запуск програмних застосувань зі сценаріїв WSH. Імітація натиснення клавіш у програмних застосуваннях 41

Лабораторна робота № 9. Введення до оболонки Windows Powershell. основи роботи у командному рядку powershell 45

Лабораторна робота № 10. Управління виведенням команд у Windows Powershell. перегляд структури, фільтрація та сортування об'єктів 56

Лабораторна робота № 11. Форматування результату виведення команд у Windows Powershell. Збереження даних у файл 65

Лабораторна робота № 12. Використання регулярних виразів у Windows Powershell 71

Лабораторна робота № 13. Програмування сценаріїв Windows Powershell. Керуючі конструкції мови Powershell: оператори перевірки умов, цикли. використання масивів 78

Лабораторна робота № 14. Робота з COM- та .NET-об’єктами у сценаріях Powershell 87

Лабораторна робота № 1. Операційна система MS-DOS. Основні поняття і команди

Інтерфейс MS DOS

MS-DOS (скор. Від англ. Microsoft Disk Operating System – дискова ОС від Microsoft) – комерційна операційна система для персональних комп'ютерів фірми Microsoft. MS-DOS – найвідоміша ОС із сімейства DOS, раніше встановлювана на більшість PC-сумісних комп'ютерів. Згодом вона була витіснена **Microsoft Windows**, і пізніше **Windows NT**.

MS-DOS була створена у 1981 р. І набула широкого поширення одночасно зі створенням комп'ютера IBM PC. Є однокористувацькою та однозадачною.

Початкове завантаження **DOS** виконується автоматично в наступних випадках:

- включення електроживлення комп'ютера;
- перезавантаження комп'ютера за допомогою кнопки **Reset**;
- перезавантаження комп'ютера за допомогою клавіш **Ctrl + Alt + Del**.

Діалог користувача з **DOS** здійснюється у формі команд, що вводяться користувачем у відповідь на запрошення **DOS**: ***C:***\>, що означає, що **DOS** готова до прийому команд. Запрошення **DOS** зазвичай містить інформацію про поточний диск і каталог.

Кожна команда користувача означає, що **DOS** повинна виконати ту чи іншу дію. Команда **DOS** складається з імені команди або програми, що викликається програми і, іноді, параметрів, розділених пробілами. Введення команди закінчується клавішею **Enter**. Регістр букв у записі команди значення не має.

Файли і каталоги

Інформація на дисках зберігається у файлах. Файл – це поіменована область на диску або іншому машинному носії. У файлах можуть зберігатися тексти програм, документи, малюнки, готові до виконання програми і т. д.

Кожен файл має позначення, яке складається з двох частин: імені (від 1 до 8 символів у **MS-DOS**, і до 254 в сучасних **ОС**) і розширення (від 1 до 3 символів). Розширення файлу є необов'язковим і, як правило, описує зміст файлу. Багато програм встановлюють розширення імені

файлу, і завдяки цьому можна дізнатися, у якій програмі був створений файл.

Приклади:

- ***.com, .exe*** – програмні (виконувані) файли;
- ***.bat*** – командні файли;
- ***.pas*** – програми на мові програмування Паскаль;
- ***.bak*** – копія файлу, що робиться перед його зміною;
- ***.jpg, .bmp*** – графічні файли;
- ***.txt*** – текстові файл;
- ***.doc, .docx –*** стандартне розширення файлів редактора Microsoft Word і т. ін.

Каталог – це спеціальне місце на диску, в якому зберігаються (реєструються) файли та інші каталоги (підкаталоги). Кожен каталог має ім'я. Якщо каталог А знаходиться в каталозі В, то А називається підкаталогом (вкладеним каталогом), а В – надкаталогом (батьківським каталогом). Зазвичай ім'я каталогу записується без розширення.

Кореневий каталог є на кожному диску і містить файли і каталоги верхнього (1-го) рівня. У каталогах 1-го рівня реєструються файли і каталоги 2-го рівня і т. ін. Таким чином, утворюється ієрархічна деревоподібна структура каталогів на диску. Поточним називається каталог, у якому зараз працює користувач (людина).

Шлях до файлу – це послідовність з імен каталогів, розділених символом **'\'**, що задає маршрут від поточного або кореневого каталогу до того каталогу, в якому знаходиться потрібний файл. Якщо шлях починається з символу **'\'**, то маршрут обчислюється від кореневого каталогу, інакше – від поточного каталогу.

Логічні диски. Як правило, жорсткий диск комп'ютера (вінчестер) для зручності роботи користувача за допомогою спеціальної програми розбивають на декілька частин – логічних дисків, з якими потім можна працювати як з окремими дисками. Кожен логічний диск має своє позначення – **C:, D:**. Імена **A:** і **B:** зазвичай зарезервовані для дискет, а нумерація логічних дисків починається з **C:**.

Повне ім'я файлу має вигляд: ***[диск:][шлях]ім'я_файлу***. Якщо диск не вказано – мається на увазі поточний. За допомогою повного імені файлу вказується необхідний для роботи файл.

Символи ***** і **?**. У деяких командах і в іменах файлів можна вживати символи ***** і **?**. Символ ***** позначає будь-яке число будь-яких символів в імені файлу або в його розширенні. Символ **?** позначає один довільний символ або відсутність символу в імені файлу або в його розширенні.

Атрибути файлів у DOS

Для кожного файлу відповідний йому запис в каталозі містить його ім'я, дату і час останньої модифікації, а також атрибути файлу. Для файлів передбачено 4 атрибути:

1. ***Тільки для читання (Read only)*** – цей атрибут оберігає файл від зміни або видалення;
2. ***Архівний (Archive)*** – атрибут встановлюється під час створення або модифікації файлу і скидається програмами резервного копіювання для позначення того, що копія файлу поміщена в архів;
3. ***Прихований (Hidden)*** – приховані файли не видно в команді **DOS *dir*** і в програмі **FAR**.
4. ***Системний (System)*** – атрибут використовується деякими системними файлами **DOS**, іноді в сукупності з атрибутом «прихований». Кожен з цих атрибутів може бути встановлений або ні. Як правило, більшість файлів має встановлений тільки атрибут «архівний».

Деякі команди DOS

1. ***Перегляд (зміна) системної дати***

Для виведення інформації про дату і установки дати в комп'ютері є команда ***date***. Формат команди:

date [mm-dd-yy]

У квадратних дужках вказується необов'язковий параметр. Підказка **DOS** вкаже, у якому порядку треба вводити дату (**mm** – місяць, **dd** – день, **yy** – рік). У цьому разі новою датою будуть позначені всі створені згодом файли. Якщо ви не хочете змінювати дату, встановлену в комп'ютері, натисніть **Enter**. Команду можна застосовувати будь-яку кількість разів.

Існує два способи застосування команди:

- ***date 10-02-2017***
- ***date***

У процесі цього з'явиться повідомлення:

Current date is: Mon 1-12-2017
Enter new date (mm-dd-yy):

2. ***Перегляд (зміна) системного часу***

Для виведення інформації про системний час і установки часу в комп'ютері є команда ***time***. Формат команди:

time [чч: мм: сс.дд], де

- ***чч*** – годинник (0 – 23),
- ***мм*** – хвилини (0 – 59),
- ***сс*** – секунди (0 – 59),
- ***дд*** – соті частки (0 – 99).

У момент включення комп'ютера запускається його внутрішній годинник. Команда дозволяє перевіряти або змінювати час неодноразово.

Новим часом позначаються всі створені згодом файли.

Існує два способи застосування команди:

- ***time 11:15***
- ***time***

З'явиться повідомлення:

Current time is 15: 23: 05.12
Enter new time:

Частина цифр може бути опущена і тоді їм будуть присвоєні нульові значення.

3. Зміна поточного диска

Для зміни поточного диска треба набрати ім'я диска, який повинен стати поточним, і потім двокрапку, наприклад:

C: – перехід на диск **C:**

4. Зміна поточного каталогу

Для зміни поточного каталогу є команда ***cd*** (Change Directory). Формат команди:

cd [диск:] шлях

Якщо заданий диск, то поточний каталог змінюється на цьому диску, інакше – на поточному диску. Команда ***cd*** без параметрів повідомляє поточні диск і каталог.

Приклади:

- ***cd ..*** – перехід на один рівень вище поточного каталогу;
- ***cd *** – перехід в кореневий каталог поточного диска.

5. Перегляд каталогу

Для виведення змісту каталогу є команда ***dir***. Формат команди:

dir [диск:][шлях \][ім'я_файлу][/опції]

В імені файлу можна вживати символи ******* і ***?***.

Якщо ім'я файлу не задано, то виводиться весь вміст каталогу. Інакше виводяться відомості тільки про цей файл або групи файлів. Для кожного файлу команда повідомляє його ім'я, розширення, розмір в байтах, дату і час створення або останнього оновлення. Підкаталоги відзначаються **<*DIR*>**. У кінці виведення повідомляється підсумкова інформація щодо кількості виведених файлів і каталогів, загального обсягу виведених файлів і розміру вільного простору на диску. Якщо в

команді не вказані диск або шлях, то мається на увазі поточний диск і поточний каталог.

Опції:

- ***/P*** – поекранне (посторінкове) виведення вмісту.
- ***/W*** – виведення інформації тільки про імена файлів і каталогів.
- ***/O:N*** – сортування за іменем файлу.
- ***/O:E*** – сортування за розширенням файлу.
- ***/O:S*** – сортування за розміром файлу (за зростанням).
- ***/S*** – виводиться зміст (або відомості про зазначені файли) не тільки для зазначеного в команді каталогу, але й для всіх його підкаталогів.

Приклади:

dir – вивести зміст поточного каталогу;

dir *.exe – вивести відомості про всі файли з розширенням .exe з поточного каталогу;

dir c:\nc – вивести зміст каталогу ***NC*** на диску ***C:***.

Завдання

1. Вивести поточні системні дату та час.
2. Перейти на кореневий диск ***C:***.
3. Вивести вміст диска ***C:*** у посторінковому режимі.
4. Вивести вміст каталогу ***TEMP*** диска ***C:*** у посторінковому режимі, відсортований за іменами файлів.
5. Перейти в каталог ***C:\PROGRAM FILES\<Деяка_папка>*** (на Ваш розсуд). Скількома способами можна здійснити таку операцію?
6. Вивести зміст поточного каталогу.
7. Вивести відомості про файли з розширенням ***TXT*** з поточного каталогу.
8. Вивести інформацію про файли поточного каталогу, що мають ім’я довжиною не більше 5 символів.
9. Вивести зміст каталогу ***PROGRAM FILES*** на диску ***C:***.
10. Перейти на кореневий диск ***C:***
11. Вивести інформацію про файли каталогу ***<Деяка_папка>*** диска ***C:***, що починаються на літеру ***F***.
12. Вивести вміст диска ***C:*** в короткому режимі, відсортований за розширенням файлів.

Контрольні питання

1. Охарактеризуйте поняття файлу та каталогу.
2. Що таке кореневий, поточний та батьківський каталоги та підкаталоги?
3. Поясніть, що таке запрошення DOS і його командний рядок.
4. Які основні команди загального призначення ви знаєте?
5. Як здійснити зміну поточного каталогу та диска?
6. Як здійснюється перегляд каталогів та дерева каталогів?

Лабораторна робота № 2. Створення і видалення файлів і каталогів у MS-DOS

Теоретичні відомості

Створення текстових файлів

Для створення текстових файлів використовується команда ***copy con***.

Формат команди:

copy con [диск:][шлях\]ім'я_файлу.

Після введення цієї команди потрібно послідовно вводити рядки файлу, закінчуючи введення кожного рядка клавішею **Enter**. Після введення останнього рядка потрібно натиснути клавішу **F6** або **Ctrl+Z** (комп'ютерний код «кінець файлу»), а потім – **Enter**. Після цього з'явиться повідомлення ***1 file(s) copied*** (один файл скопійований), і на диску з'явиться файл з вказаним ім'ям.

Наприклад,

copy con myfile.txt – створення файлу ***myfile.txt*** у поточному каталозі.

Видалення файлів

Для видалення файлів використовується команда ***del***.

Формат команди:

del [диск:][шлях\]ім'я_файлу.

В імені файлу можна вживати символи ******* і ***?*** для видалення декількох файлів.

Наприклад,

del *.* – видалення всіх файлів з поточного каталогу. Після такої команди DOS запитає вас ***«Продовжити? [Y / N]»***.

Команда ***del*** не видаляти файли з атрибутом **«тільки для читання»**. Не видаляються також файли, у яких є хоча б один з атрибутів **«прихований»** і **«системний»**, а атрибут **«архівний»** не встановлено.

Наприклад:

- ***del C:\text\myfile.txt*** – видалення файлу ***myfile.txt*** з каталогу ***text*** диска ***C:***.
- ***del *.bak*** – видалити всі файли з типом ***.bak*** з поточного каталогу.

Виведення файлу на екран

Для виведення файлу на екран служить команда ***type***. Формат команди:

type ім'я_файлу

Символи ******* і ***?*** тут неприйнятні. На екран може бути виведений тільки файл з текстовою інформацією.

У випадку довгих файлів текст вводиться у нижній рядок екрана і витісняється угору допоки не буде досягнутий кінець файлу. Виведення тексту на екран (скролінг) можна призупинити натисканням ***Ctrl+S***. Повторне натискання цих клавіш відновлює виведення на екран.

Для виведення довгих файлів можна використовувати команду порядкового виведення вмісту файлу ***copy ім'я_файлу | more***.

Приклади:

- ***type myfile.txt*** – висновок на екран файлу ***myfile.txt*** з поточного каталогу.
- ***copy myfile.txt con*** – виведення файлу на екран за допомогою команди ***copy***.

Створення каталогу

Для створення нового каталогу є команда ***md*** (Make Directory). Формат команди:

md [диск:]шлях

Таким чином, можна створювати скільки завгодно підкаталогів у каталогах. Приклади:

- ***md direct*** – створення підкаталогу ***direct*** у поточному каталозі;
- ***md C:\work*** – створення каталогу ***work*** на кореневому диску ***C:***.

Знищення каталогу

Для видалення каталогу є команда ***rd*** (Remove Directory). Формат команди:

rd [диск:]шлях.

Видалити можна тільки порожній каталог. За допомогою цієї команди можна знищити поточний підкаталог і кореневий каталог.

Наприклад:

- ***rd direct*** – видалення підкаталогу ***direct*** з поточного каталогу;
- ***rd A:\text\dict*** – видалення підкаталогу ***dict*** з каталогу ***text*** на диску ***A:***.

Додаткові команди MS DOS

1. ***cls*** – служить для очищення екрана. Екран очищується від повідомлень операційної системи, у першому рядку з'являється запрошення **DOS**;
2. ***ver*** – видача номера версії **ОС**, що діє на комп'ютері;
3. ***vol*** – перевірка імені диска (мітки тома). Формат команди: ***vol [диск:]***. Якщо дисковод не вказано – береться уваги поточний дисковод.

Завдання

1. На диску N: створіть каталог з ім'ям ПРІЗВИЩЕ (прізвище студента).
2. Увійти в створений каталог.
3. Створити підкаталог ІМ'Я (ім'я студента) у поточному каталозі.
4. Увійти в створений підкаталог.
5. Створити два текстові файли text1 і text2 з довільною інформацією.
6. Вивести створені файли на екран послідовно.
7. Видалити файли. Можливі способи виконання цієї операції.
8. Видалити створені каталоги ПРІЗВИЩЕ і ІМ'Я.
9. Зайти в каталог Program files на диску C :.

Наступні пункти виконати, перебуваючи в каталозі Program files.

10. Створити каталог WORK у кореневому каталозі диска N:
11. Створити файл text.txt в створеному каталозі.
12. Продивитися вміст каталогу WORK.
13. Видалити каталог WORK.
14. Вивести номер версії операційної системи.
15. Вивести на екран мітку поточного диска.

Контрольні питання

1. Яка команда MS DOS служить для створення каталогів?
2. Як створити у командному рядку новий текстовий файл?
3. Як переглянути вміст файлу?
4. Чим буде відрізнятися результат виконання команд COPY CON file1.txt та COPY file1.txt CON?
5. Назвіть призначення команди DEL.
6. Як видалити каталог?
7. Для чого призначена команда VOL?

Лабораторна робота № 3. Копіювання і перейменування файлів і каталогів у MS-DOS

Копіювання файлів

Для копіювання файлів є команда ***copy***. Формат команди:

copy [диск:][шлях\][старе-ім'я_файлу] [диск:][шлях\][нове-ім'я_файлу] [/ V].

З каталогу, зазначеного у першому параметрі команди, копіюються файли, задані в першому параметрі команди. Диск і шлях у другому параметрі команди вказують каталог, в який копіюються файли. Якщо у другому параметрі ім'я файлу відсутнє, то імена файлів під час копіювання не змінюються.

В іменах файлів можна використовувати символи ***** і ***?***, тобто за допомогою однієї команди можна скопіювати декілька файлів. Символи ***** і ***?*** в імені файлу в другому параметрі команди вказують, що відповідні символи в іменах вихідних файлів під час копіювання не змінюються.

Приклади:

- ***copy text2.txt text3.txt*** – копіювання файлу text2.txt у поточному каталозі у файл text3.txt у поточному каталозі;
- ***copy c:*.**** – копіювання всіх файлів з кореневого каталогу диска з: у поточний каталог;
- ***copy \tt*.doc c:*.txt*** – копіювання всіх файлів з розширенням ***.doc*** з каталогу ***tt*** поточного кореневого диска на кореневий диск ***C:***, у цьому разі файли отримують розширення ***.txt.***

Якщо файл з таким самим ім'ям, як у створюваної копії, вже існує, то він оновлюється без попереджень. Команда ***copy*** не копіює файли з атрибутом «прихований».

Перевірка скопійованих файлів

За допомогою **MS-DOS** можна впевнитися у тому, що файли скопійовані на диск правильно. Перевірка може бути виконана двома способами.

Перший: перевірка полягає в тому, що в кінці команди копіювання вводиться опція ***/V***. У цьому випадку, всі файли після копіювання зчитуються і порівнюються з оригіналом. У разі виявлення дефектних зон у файлі-копії виводиться відповідне повідомлення і можна повторити спробу копіювання.

Другий: пов'язаний з попереднім (до команди ***copy***) застосуванням команди ***verify*** (звірка інформації). Формат команди:

verify [on] або ***verify [off].***

За командою ***Verify ON***, **DOS** звіряє всю інформацію, яку записують на диск. Після запису кожен блок зчитується, інформація звіряється з вихідною, і тільки тоді **DOS** продовжує роботу. Для припинення звірки, вводиться команда ***Verify OFF***. Якщо ви не знаєте, у якому стані знаходиться процес звірки – наберіть команду ***Verify*** – у цьому випадку на екран виводиться стан звірки: ***ON*** – виконується, ***OFF*** – припинена.

Обмін даними між периферійними пристроями (ПП)

Команда ***copy*** може застосовуватися не тільки для копіювання, але й для реалізації обміну між ПП. Ця команда дозволяє пересилати інформацію з клавіатури на диск, з диска на принтер і т. ін. У команді ***copy*** замість імен файлів можна використовувати позначення пристроїв, наприклад:

con – консоль (клавіатура для введення, монітор для виведення).

prn або ***lpt1*** – основний принтер системи (тільки як вихідний файл).

Приклади:

copy paper.txt prn – виведення файлу ***paper.txt*** на принтер;

copy paper.txt con - виведення файлу ***paper.txt*** на екран.

Об'єднання (конкатенація) текстових файлів.

У процесі копіювання можна модифікувати кінцеві показники, тобто виробляти об'єднання (конкатенацію) декількох файлів в один. Спочатку в команді вказуються через «+» імена файлів, що об'єднуються, а потім ім'я результуючого файлу. Вихідні файли не змінюються і записуються послідовно в новий файл. Якщо ім'я нового файлу не вказано – він отримує ім'я першого файлу зі списку файлів, що об'єднуються.

Файли можна вказувати за допомогою символів «*******» і «***?***», але при цьому неможливо змінювати послідовність файлів, оскільки об'єднання виконується в порядку розміщення їх на диску.

Приклади:

- ***copy file1.txt+file2.txt file3.txt*** – об'єднання файлів з іменами ***file1.txt*** і ***file2.txt*** в файл з ім'ям ***file3.txt.***
- ***copy *.doc final.doc*** – всі файли з типом ***.doc*** об'єднуються у файл ***final.doc***.
- ***copy *.lst+*.ref text.txt*** – об'єднання файлів з типами ***.lst*** і ***.ref*** у файл ***text.txt.***

Перейменування файлів

Для перейменування файлів (каталогів) є команда ***ren*** (Rename). Формат команди:

ren [диск:][шлях\] старе_ім'я_файлу нове_ім'я_файлу.

Команда дозволяє змінити ім'я наявного файлу без зміни його вмісту. Перше ім'я файлу задає ім'я файлу, що перейменовується, друге – нове ім'я файлу. Диск і шлях задають, у тому каталозі, в якому перейменовуються файли. Якщо диск і шлях не вказані – береться до уваги поточний диск і поточний каталог. У новому імені файлу диск і шлях не вказується.

В іменах файлів можна вживати символи ******* і ***?***. Якщо символи ******* і ***?*** є у другому імені файлу в команді, то символи імен файлів на відповідних позиціях не змінюються.

Приклади:

- ***ren dir1\text1.txt text2.txt*** – перейменування файлу ***text1.txt*** у файл з новим ім'ям ***text2.txt*** у каталозі ***dir1***;
- ***ren text3.txt text4.txt*** – перейменування файлу ***text3.txt*** у файл з новим ім'ям ***text4.txt*** у поточному каталозі;
- ***ren a:*.doc *.tx***t – перейменування всіх файлів з розширенням ***.doc*** кореневого диска ***A:*** у файли з колишніми іменами, але з розширенням ***.txt.***

Команда ren не виконує жодних перейменувань файлів з атрибутом «прихований».

Завдання

1. На диску ***H:*** створити каталог ***dir1***.
2. Увійти в створений каталог.
3. Створити підкаталог ***dir2***.
4. Створити текстовий файл в каталозі ***dir1***.
5. Перейменувати створений файл.
6. Скопіювати файл у підкаталог ***dir2*** з тим самим іменем.
7. Скопіювати файл з підкаталогу ***dir2*** в каталог ***dir1*** з новим іменем.
8. Вивести на екран вміст будь-якого файлу командою ***copy***.
9. Об'єднати файли каталогу ***dir1*** в новий текстовий файл.
10. Вийти зі свого каталогу.
11. Створити каталог з іменем ***dir3*** на диску ***H***.
12. Скопіювати всі файли каталогу ***dir1*** у каталог ***dir3***. Скількома способами можна виконати таку операцію?
13. Скопіювати групу файлів з розширенням ***.txt*** з каталогу ***C:\Program Files\Far*** в каталог ***dir3***, використовуючи перевірку операції копіювання.

14. З'ясувати стан звірки інформації. Як його змінити на протилежний?
15. Змінити у всіх файлів з типом ***.txt*** в каталозі ***dir1*** тип на ***.tmp***.
16. Скопіювати в каталог ***dir2*** всі файли каталогу ***dir1***. Як зміниться командний рядок залежно від поточного каталогу ***dir1***, ***dir2***, ***temp***, ***H:***\?
17. Знищити створені каталоги і файли.

Контрольні питання

1. Які команди називаються внутрішніми командами **MS-DOS**?
2. Які команди називаються зовнішніми командами **MS-DOS**?
3. Які символи заміщення можна використовувати в командах **MS-DOS**?
4. Що означає команда ***A:\>DIR ?AC.TXT?***
5. Як розуміти команду ***C:\FOTON\> DIR *.FT?***
6. Як здійснити виведення каталогу на друк?
7. Що означає команда ***C:\> COPY *.TXT B:\USER***?
8. Яким способом можна змінити ім'я файлу?
9. У чому полягають відмінності команди ***COPY*** від команди ***DISKCOPY***?

Лабораторна робота № 4. Програмування командних BAT-файлів, обробка аргументів, мітки, оператори переходу і розгалуження, організація циклів

Пакетні командні файли (ПКФ) в системі MS-DOS

Пакетні командні файли використовуються в операційних системах для того, щоб мати можливість задавати послідовність команд ОС, управляти запуском завдань, створювати різні варіанти робочого середовища.

ПКФ в системі MS-DOS є текстовими файлами в коді ASCII, які містять послідовності команд ОС. Вони повинні мати розширення ***.bat*** і мати ім'я не більше восьми символів. Але у поточному каталозі не повинно бути однойменних файлів з розширенням ***.com*** або ***.exe***.

В ОС MS-DOS спеціально для ПКФ передбачено вісім команд і спеціальний символ **@** (табл.1). Вони є внутрішніми командами ОС, розміщуються в командному процесорі і доступні з будь-якого каталогу.

Таблиця 1

Команда	Призначення
@	Не виводить на екран дисплея рядок, який слідує за ним
call	Здійснює виконання одного пакета команд всередині іншого
cls	Очищення екрана
echo off	Забороняє виведення на екран командних рядків
echo on	Дозволяє виведення на екран командних рядків
echo <текст>	Виведення на екран текстової інформації (з дією команди ***echo off***)
for	Завдання циклу
goto <мітка>	Передача управління на вказану мітку
if	Умовне виконання команди
pause	Переривання виконання програми
rem	Виведення тексту на екран дисплея
shift	Зрушення ліворуч параметрів команди

Для запуску ПКФ вводиться його ім'я без розширення. У разі, якщо ПКФ знаходиться не в поточному каталозі, шлях доступу до ПКФ повинен бути визначений командою ***path***. Якщо у якій-небудь команді пакета була допущена помилка, то виконання пакету припиняється і на екран виводиться повідомлення про помилку.

Можна перервати виконання пакету натисканням **Ctrl-C**. У цьому випадку виводиться повідомлення:

Terminate batch job (Y / N)?.

(Завершити роботу командного файлу (Так / Ні)?.

ПКФ можна сформувати за допомогою будь-якого текстового редактора або інших програм, що дозволяють створювати файли в коді ASCII.

Дія команд у командних файлах

Розглянемо більш докладно деякі команди в ПКФ.

1. ***echo on/off.***

ECHO <текст повідомлення> – вкл/викл режиму виведення команди перед її виконанням.

@ – якщо цей символ поставлений у початок рядка, то наступна команда не виводиться за включеного режиму виведення команди перед її виконанням (тобто якщо ***echo on***).

2. ***goto.***

Після імені цієї команди повідомляється мітка командного рядка, до якої виконується перехід. Якщо перед рядком зазначена двокрапка, то це означає, що за ним слідує мітка, а не команда. Між двокрапкою і іменем пробіл не ставиться, між іменем і командою ***goto*** залишають пробіл. Довжина мітки обмежується 8 символами.

Наприклад,

goto exit *оператор безумовного переходу за міткою*

:exit.

3. ***choice [/c:список символів] [/t:символ, число секунд] повідомлення***

/c:список символів – вказує допустимі символи, які може ввести користувач у відповідь на повідомлення. Якщо цей параметр не вказано, допустимі символи – ***y*** і ***n.***

/t:символ, число секунд – якщо цей параметр вказаний, то в разі, коли користувач автоматично через певний час не натиснув на жодну клавішу, приймається відповідь ***символ.***

Наприклад:

choice /c:yn «запустити програму alfa?» – при відповіді ***n*** значення змінної ***errorlevel*** встановлюється рівним 2, а при відповіді ***y*** – рівним 1.

4. ***if <умова> <дія>*** – оператор умови.

Наприклад:

if errorlevel 1 goto exit.

<умова> може мати один з чотирьох форматів:

- ***<набір_символів_1> == <набір_символів_2>.***

Виконується, якщо ***<набір_символів_1>*** і ***<набір_символів_2>*** є ідентичними після заміщення параметрів.

- ***errorlevel <номер>.***

Виконується, якщо для останньої команди, яку виконала ОС, код завершення дорівнює або більше значення ***<номер>.***

- ***exist <ім'я файлу>.***

Виконується, якщо існує файл з вказаним ім'ям.

- ***not <умова>.***

Виконується, якщо задана умова хибна.

5. ***command /y /c імя_командного_файлу [параметри]*** – виконання командного файлу в покроковому режимі. Текст кожної команди буде виводитися перед виконанням на екран. Для виконання команди треба натиснути клавішу **Y** або **Enter**.

6. ***pause.***

Виводить на екран повідомлення: ***Strike a key when ready ....*** І після натискання клавіші далі виконується командний файл. У період паузи можливий вихід з виконання командного файлу шляхом натискання клавіш ***Ctrl + C*** або ***Ctrl + Break***.

7. ***shift.***

Формат команди простий: ***shift***. У ПКФ можна використовувати до десятка параметрів, за допомогою цієї команди цю кількість можна збільшити. Команда ***shift*** привласнює нові значення параметрам ***%0*** – ***%9***. Параметру ***%0*** присвоюється старе значення ***%1***, ***%1*** – старе значення ***%2*** і т. ін. ***%9*** присвоюється значення, що випливає за старим значенням ***%9*** у командному рядку.

8. ***for.***

for %%n in (група) do команда %%n, де:

- ***N*** – однобуквена змінна, що послідовно приймає значення, що входять до параметру ***(група)***;
- ***(група)*** – ім'я або імена файлів, можуть зазначатися абсолютним шляхом; ***команда*** – команда ОС **MS-DOS**.

9. ***call.***

call [<дисковод:>] [<шлях_доступу>] <командний_файл> <параметри>, де

- ***<дисковод:>*** – дисковод, диск якого містить командний файл;
- ***<шлях_доступу>*** – шлях доступу до цього **ПКФ**;
- ***<командний_файл>*** – ім'я **ПКФ** без розширення.

Завдання

1. Написати командний файл, який буде копіювати з поточного каталогу всі файли з розширенням txt, крім одного файлу, вказаного у якості другого параметра командного рядка, в каталог, вказаний першим параметром. Якщо ім'я каталогу, в який повинно проводитися копіювання, не задано, то вивести повідомлення про це, якщо буде введений символ Y – продовжити роботу, тобто створити каталог і скопіювати файли, N – перервати виконання файлу.

Якщо пакетний файл запускається взагалі без параметрів, то вивести опис його синтаксису.

2. Створити командний файл, який виводив би вміст каталогу, зазначеного у якості параметра командного рядка, причому користувачеві має бути надана можливість вибору за допомогою меню пристрою для виведення: на екран (інформація виводиться на одному екрані), у текстовий файл catalog.txt .

Командний файл повинен обробляти два ключа:

- ***/A*** – сортування виведеної інформації за алфавітом;
- ***/D*** – за датою створення.

Якщо пакетний файл запускається взагалі без параметрів, то необхідно вивести опис його синтаксису.

3. Створити пакетний файл, який розкладає вміст папки, ім'я якої передається у якості параметра командного файлу, у підпапки відповідно до розширення файлів – всі***.exe*** файли – в папку ***EXE***, ***.txt*** – у ***TXT*** і т. ін. Створити окрему папку для файлів без розширення.

Контрольні питання

1. Що таке командний файл?
2. Який файл є командним інтерпретатором?
3. Які команди операційної системи можуть використовуватися в **ПКФ?**
4. Як запускаються **ПКФ?**
5. Що станеться у процесі виконання **ПКФ,** якщо в командах командного файлу допущена помилка?
6. Яким чином можна перервати виконання пакету?
7. Яким чином організовується розгалуження у **ПКФ?**
8. Скільки параметрів можна використовувати під час виклику **ПКФ?**
9. Як діє команда ***call***?
10. Яке призначення символу **@**?

Лабораторна робота № 5. Основи програмування скриптів WSH. Основні методи об'єкта WScript. Запис інформації у документи Word та Excel зі скрипта

Windows Script Host (WSH) являє собою технологію автоматизації вирішення завдань моніторингу та управління ресурсами в ОС сімейства Microsoft. **WSH** прийшов на зміну мови командних файлів (bat-файлів), вирішуючи аналогічні завдання, але з більш потужною функціональністю.

Сервер сценаріїв **WSH** є розвиненим інструментом, що надає єдиний інтерфейс (об'єктну модель) для спеціалізованих мов, основними з яких є **VBScript**, **Jscript**, але допускається використання додаткових мов, таких як **PerlScript**, **TCL**, **Python** і т. ін. Основу функціональних можливостей технології **WSH** складають компоненти **ActiveX**, сервісні функції яких дозволяють керувати ресурсами системи (створювати ярлики програм, вимикати комп'ютер, змінювати записи в реєстрі, працювати з мережею і користувачами, виводити список дисків, підключати/відключати мережеві диски, отримувати ім'я комп'ютера і користувача, працювати зі змінними оточення, видавати діалогові та інформаційні повідомлення тощо), а також керувати роботою інших додатків (серверів автоматизації), наприклад, Microsoft Word і чи Micosoft Excel.

У цьому разі для роботи з **WSH** використовуються мови програмування **VBScript** і **JScript**, підтримка яких вбудована в операційну систему і не вимагає налаштування спеціалізованих систем програмування. Середовищем програмування фактично стає текстовий редактор, наприклад, стандартний **Блокнот**. Типовий WSH-сценарій – це звичайний текстовий файл з розширенням ***.js*** (для **JScript**) або ***.vbs*** (для **VBScript**) відповідно.

Мова **JScript** – скриптова, об'єктно-орієнтована мова програмування, що інтерпретується, розроблена фірмою Microsoft. Спочатку вона використовувалася для створення динамічних HTML-сторінок в рамках технології ASP. Синтаксично вона дуже близька до мов **Java** і **C**, разом з тим є хорошим вибором для написання скриптів тими програмістами, які володіють ними. Мова **VBScript (Visual Basic Script Edition)** – є полегшеною версією мови **Microsoft Visual Basic**, і буде хорошим вибором для тих, хто володіє **Visual Basic** або **VBA (Visual Basic for Application).**

Для запуску WSH-скрипта використовуються команди ***cscript.exe*** і ***wscript.exe***. Перша з них направляє виведення скрипта на стандартну консоль виведення в текстовий режим, а друга – в діалогове вікно. У цьому можна переконатися, якщо запустити наступний скрипт на мові **VBScript**.

```
'Найпростіший скрипт на VBScript.
WScript.Echo «Привіт!».
```

командами.

cscript.exe hello.vbs та ***wscript.exe hello.vbs***.

Аналогічний за функціональністю скрипт на **JScript** виглядає наступним чином:

```
//Найпростіший скрипт на JScript.
WScript.Echo(«Привіт!»);
```

Під час запуску сценарію з командного рядка можна визначити параметри хоста (див. табл. 2), які включають або виключають різні опції **Windows Scripting Host** і завжди передуються двома слешами (//).

Таблиця 2

Параметри командного рядка для cscript.exe

Параметр	Опис
//B	Включає пакетний режим. При цьому на екран будуть виводитися всі повідомлення про помилки в сценарії
//T:nn	Задає таймаут в секундах. Сценарій виконується не більше nn секунд
//I	Вимикає пакетний режим (використовується за замовчуванням). На екран не виводяться повідомлення про помилки
//logo	Включає у висновок скрипту напис про версії та розробника WSH (використовується за замовчуванням)
//nologo	Пригнічує інформацію про версії та розробника **WSH**
//H: Cscript or Wscript	Робить **CSCRIPT.EXE** або **WSCRIPT.EXE** хостом за замовчуванням, тобто асоціює їх з розширеннями скриптів
//S	Зберігає налаштування командного рядка для поточного користувача
//D	Функція переходу в режим Active Debuging
//X	Виконання сценарію у відлагоджувачі

Безсумнівною перевагою WSH порівняно з командними файлами є підтримка об'єктно-орієнтованої парадигми, коли основні об'єкти предметної області (такі як файл на диску, ярлик на робочому столі, змінна оточення, запущений процес і т. ін.) представлені програмними об'єктами. Маніпулювання властивостями і методами цих об'єктів дозволяє швидко і ефективно отримати або змінити його властивості.

За допомогою внутрішніх об'єктів WSH зі сценаріїв можна виконувати основні завдання:

- виводити інформацію в стандартний вихідний потік (на екран) або у вікно Windows;
- читати інформацію зі стандартного вхідного потоку (тобто вводити її з клавіатури) або використовувати інформацію, виведену іншою командою;
- використовувати властивості і методи зовнішніх об'єктів, а також обробляти події;
- запускати нові процеси або активізувати вже наявні;
- працювати з локальною мережею: визначати ім'я зареєстрованого користувача, підключати мережеві диски та принтери;
- переглядати і змінювати змінні середовища;
- отримувати доступ до спеціальних папок Windows; створювати ярлики Windows;
- працювати з системним реєстром.

До WSH входять наступні об'єкти:

1. ***WScript***. Це головний об'єкт WSH, який служить для створення інших об'єктів або зв'язку з ними, містить відомості про сервер сценаріїв, а також дозволяє вводити інформацію з клавіатури і виводити її на екран або у вікно Windows;
2. ***WshArguments***. Забезпечує доступ до параметрів командного рядка за-пущеного сценарію або ярлика Windows;
3. ***WshEnvironment***. Призначений для роботи зі змінними середовища;
4. ***WshSpeciaiFoiders***. Забезпечує доступ до спеціальних папок Windows;
5. ***WshNetwork***. Використовується під час роботи з локальною мережею: містить мережеву інформацію для локального комп'ютера, дозволяє підключати мережеві диски та принтери;
6. ***WshShell***. Дозволяє запускати процеси, створювати ярлики, працювати зі змінними середовища, системним реєстром і спеціальними папками Windows;
7. ***WshShortcut***. Дозволяє працювати з ярликами Windows;
8. ***WshUriShortcut***. Призначений для роботи з ярликами мережевих ресурсів.

Крім цього, є об'єкт ***Scripting.FileSystemObject***, що забезпечує доступ до файлової системи комп'ютера.

Одне з центральних місць у скриптах **WSH** займає об'єкт ***WScript***. Об'єкт ***WScript*** можна використовувати в сценарії **WSH** відразу, без будь-якого попереднього опису або створення, тому що його екземпляр створюється сервером сценаріїв (***CScript.exe*** або ***WScript.exe***) автоматично. Об'єкт ***WScript*** дозволяє здійснювати операції введення-виведення інформації, отримувати список аргументів скрипта, отримувати інформацію про сервер **WSH**, а також дозволяє скриптам запускати додатки і керувати ними.

Таблиця 3

Властивості та методи об'єкта WScript

Властивість	**Опис**
Arguments	Повертає покажчик на колекцію ***WshArguments*** аргументів командного рядка
FullName	Повертає ім'я виконуваного файлу хоста і повний шлях до нього (наприклад, ***C:\Windows\ wscript.exe***)
Name	Виводить назву сервера об'єкта: **Windows Scripting Host**
Path	Визначає каталог і шлях, що містять ***wscript.exe*** або ***cscript.exe***
ScriptFullName	Повертає повний шлях і ім'я виконуваного в цей момент скрипта
ScriptName	Те саме, що і ***ScriptFullName***, але без шляху
StdErr	Дозволяє запущеному сценарію записувати повідомлення в стандартний потік для помилок
StdIn	Дозволяє запущеному сценарію читати інформацію зі стандартного вхідного потоку
StdOut	Дозволяє запущеному сценарію записувати інформацію в стандартний вихідний потік
Version	Повертає версію встановленого **Windows Scripting Host**
Метод	**Опис**
CreateObject	Створює об'єкт за його **ProgID**
ConnectObject	Дозволяє підключитися до подій об'єкта. Як параметр приймає об'єкт, до якого потрібно підключитися, і префікс відповідних подій процедур, реалізованих у скрипті
DisconnectObject	Вимикає від об'єкта, підключеного попереднім методом
Echo	Виводить текстовий рядок (в ***cscript*** – в **StdOut**, в ***wscript*** – у вигляді діалогового вікна)
GetObject	Дозволяє отримати покажчик на об'єкт з файлу або об'єкта, наданого в секції **strProgID**
Quit	Завершує скрипт
Sleep	Переводить скрипт в неактивний стан на час, вказаний у мілісекундах

Необхідно відзначити, що стандартні потоки введення-виведення доступні в об'єкті ***WScript*** через властивості ***StdIn***, ***StdOut*** і ***StdErr***, тільки якщо сценарій запускався в консольному режимі за допомогою ***cscript.exe***.

Розглянемо простий скрипт на мові JScript, що використовує властивості і методи об'єкта ***WScript***:

```
var args;
args=WScript.Arguments;
for (var i=0; i<args.length; i++) //можна i<args.Count()
  WScript.Echo(«Аргумент «+i+»: «+args(i));
```

Запустити цей скрипт (припустимо, що він записаний у файл ***arglist.js***) можна з командного рядка:

```
cscript.exe arglist.js /a myFolder *.txt
```

Можна запустити і скрипт з віконним інтерфейсом:

```
wscript.exe arglist.js /a myFolder *.txt
```

але кожен виклик методу ***Echo*** буде створювати окреме вікно повідомлення.

Якщо встановлена за замовчуванням асоціація розширення ***js*** (і ***vbs***) з програмою ***cscript.exe*** не була змінена, то скрипт запускається простіше:

```
arglist.js /a myFolder *.txt
```

Якщо створений консольний скрипт, то для введення-виведення можна використовувати методи ***Write, WriteLine, Read, ReadLine, ReadAll, Skip, SkipLine*** властивостей-об'єктів ***StdIn, StdOut і StdErr***. Наступний приклад запитує у користувача повне ім'я файлу і перевіряє, чи існує він на диску:

```
var FileName, fso;
WScript.StdOut.WriteLine(«Введіть повне ім'я файлу «);
FileName=WScript.StdIn.ReadLine();
fso = WScript.CreateObject(«Scripting.FileSystemObject»)
if (fso.FileExists(FileName))
  WScript.StdOut.WriteLine(«Такий файл вже існує «);
else
  WScript.StdOut.WriteLine(«Такий файл не існує «);
```

Виведення інформації зі скрипта у вихідний потік дозволяє перенаправляти інформацію у файл або організовувати конвеєр стандартними операторами операційної системи (>, >>, |):

```
cscript.exe someScript.js | find “2016” > lastYear.txt
```

З розглянутих прикладів очевидно, що головний секрет успішного застосування **WSH** полягає в знанні та вмінні маніпулювати властивостями і методами перерахованими в таблиці 1 об'єктів. Деякі з об'єктів створюються автоматично в ході ініціалізації властивості інших об'єктів (наприклад, вже розглянутий нами раніше ***WshArguments***), інші необхідно створювати в коді. Для цього служить метод ***CreateObject*** об'єкта ***WScript***. Наприклад, для створення об'єкта ***WshShell*** необхідно виконати:

var wh=WScript.CreateObject(«WScript.Shell»);

Після виконання цього методу буде створений об'єкт ***WshShell***, доступ до властивостей і методів якого можливий за допомогою змінної ***wh***.

Як параметр метод ***CreateObject*** приймає програмний ідентифікатор об'єкта (**Programmic Identifier, ProgID**). **ProgID** є унікальним ідентифікатором об'єкта, його зв'язок з конкретним розташуванням файлів, що зберігають код об'єкта, здійснюється за допомогою технології **COM** через системний реєстр. Зокрема, так можна зв'язати скрипт з екземпляром **Microsoft Excel** і записати інформацію в осередку аркуша документа:

```
// створюємо об'єкт - excel-додаток
var objXL = WScript.CreateObject(«Excel.Application»);

// робимо вікно видимим і створюємо робочу книгу
objXL.Visible = true;
objXL.WorkBooks.Add;

// встановлюємо ширину першого стовпця
objXL.Columns(1).ColumnWidth = 20;

// записуємо рядок у комірку (1,1)
objXL.Cells(1, 1).Value = «Створено з WSH»;
```

Наступний приклад демонструє, як можна дописати інформацію в кінець файлу типу ***doc*** (документ **Microsoft Word**):

```
var strDoc = «c:\\test.doc»;
var d = new Date();   // отримуємо поточну дату
var strText;
strText += d.getDate() + «/»;                    // витягуємо день,
strText += (d.getMonth() + 1) + «/»;     //місяць
strText += d.getYear();                          //рік
strText += « Доданий рядок з WSH скрипта\n» ;
if (WScript.CreateObject(«Scripting.FileSystemObject»).
FileExists(strDoc))
```

```
{  var oWord=WScript.CreateObject(«Word.Application»)
   var oDoc=oWord.Documents.Open(strDoc);
   oDoc.Content.InsertAfter(strText)
   oDoc.Save();
   oDoc.Close();
   oWord.Quit();
}
else
{
   WScript.Echo(«Document [« + strDoc + «] not found»);
      WScript.Quit(1);
}
WScript.Quit(0);
```

Отримати ProgID список методів і властивостей для об'єкта, який вас цікавить можна з технічної документації або використовуючи бібліотеку його типів.

Завдання

1. Написати скрипт, що виводить текстове повідомлення методами WScript.Echo і WshShell.Popup. Запустити скрипт командами cscript і wscript, порівняти результати виведення.

2. Написати скрипт, що приймає введені від користувача рядки, створює нову книгу Excel, документ Word та записує введені користувачем рядки (комірки, у які буде записуватися інформація в Excel обрати на власний розсуд). Під час введення користувачем слова «Quit» завершити введення та зберегти обидва файли.

Контрольні питання

1. Для чого призначений об'єкт WScript?
2. За допомогою яких властивостей об'єкта WScript можна отримати доступ до стандартних потоків введення, виведення та помилок?
3. Для чого призначений метод CreateObject об'єкта WScript?
4. Для чого призначений метод ConnectObject об'єкта WScript?
5. Як зі сценарію WSH відкрити файл електронної таблиці Microsoft Excel та записати інформацію у її комірках?
6. Як зі сценарію WSH відкрити файл документу Microsoft Word та записати інформацію у ньому?

Лабораторна робота № 6. Використання об'єкту FileSystemObject для роботи з файловою системою. Методи роботи з файлами та каталогами у сценаріях WSH

Найбільшу частку виконуваних в процесі адміністрування і обслуговування комп'ютера операцій складають дії над об'єктами файлової системи: створення, читання і зміна вмісту файлів і папок, їх копіювання та видалення, отримання інформації про диски, папки, файли і т. ін. Скрипти **WSH** можуть виконувати подібні операції за допомогою об'єкта ***FileSystemObject***. Він не належить до **WSH**, а є стороннім **ActiveX**-об'єктом. Але, використовуючи метод ***CreateObject***, скрипт **WSH** може створювати подібні об'єкти і використовувати багату функціональність ***FileSystemObject***:

var FSO = WScript.CreateObject("Scripting.FileSystemObject");

Об'єкт ***FileSystemObject*** надає в розпорядження програміста об'єкти і колекції, перераховані у таблиці 4.

Таблиця 4

Об'єкти і колекції для роботи з файловою системою

Об'єкт/Колекція	Опис
FileSystemObject	**Основний об'єкт.** Містить методи і властивості, які забезпечують доступ до файлової системи комп'ютера: створюють, видаляють, отримують інформацію, і керують дисками, папками та файлами. Багато методів цього об'єкта дублю-ються в інших об'єктах (наприклад, видалити файл можна як викликом ***DeleteFile***, так і викликом методу ***Delete*** об'єкта ***File***)
Drive	**Об'єкт.** Містить методи і властивості, які дозволяють збирати інформацію про накопичу-вачі, наявні в системі: ім'я диска, мітка тому, загальний розмір і обсяг вільного місця на диску
Drives	**Колекція об'єктів типу *Drive*.** Включає в себе всі диски на цьому комп'ютері незалежно від їх типу
File	**Об'єкт.** Містить методи і властивості, які дозволяють виконувати основні операції з файлом: створювати, видаляти, або переміщувати файл, отримувати і змінювати атрибути. Його властивості зберігають характеристики файлу: розмір, ім'я, шлях до нього, атрибути

Закінчення таблиці

Files	**Колекція об'єктів типу *File*.** Містить список усіх файлів, що знаходяться у зазначеній папці
Folder	**Об'єкт.** Аналогічний ***File***, але дозволяє отримати доступ до заданої папки
Folders	**Колекція об'єктів типу *Folder*.** Містить список всіх папок, що знаходяться в заданій папці
TextStream	**Об'єкт.** Дозволяє читати і писати інформацію в текстовий файл

Для маніпулювання з файлом необхідно створити пов'язаний з ним об'єкт ***File***. Для цього призначений метод ***FileSystemObject*** (далі ***FSO***) ***GetFile***. Якщо файл не існує, він попередньо створюється методом ***FSO.CreateTextFile***. Наступний приклад демонструє застосування перерахованих методів для роботи з файлом.

```
var FSO=WScript.CreateObject(«Scripting.FileSystemObject»);
fname=«d:\\test.txt»  // можна fname=«d:/test.txt»
if (!FSO.FileExists(fname))
{
    file=FSO.CreateTextFile(«d:\\test.txt»);
    file.WriteLine(«New file»);
    file.Close ();
 }
file=FSO.GetFile(fname);
file.Copy(fname+».copy»);
//або так FSO.CopyFile(fname,fname+».copy»);
file.Delete();
```

Передостанній рядок скрипту демонструє, що для копіювання, переміщення і видалення файлів можна скористатися як методами об'єкта ***FSO***, так і об'єкта ***File*** (див. табл. 5).

Таблиця 5

Методи для роботи з файлами

Дія	Метод
Переміщення	***File.Move*** або ***FileSystemObject.MoveFile***
Копіювання	***File.Copy*** або ***FileSystemObject.CopyFile***
Видалення	***File.Delete*** або ***FileSystemObject.DeleteFile***
Створення	***FSO.CreateTextFile***

Набір властивостей об'єкта ***File*** надає програмісту доступ до основних характеристик файлу (табл. 6).

Таблиця 6

Властивості об'єкта File

Властивість	Опис
Attributes	Дозволяє переглянути або встановити атрибути файлу
DateCreated	Містить дату та час створення файлу. Доступна тільки для читання
DateLastAccessed	Містить дату та час останнього звернення. Доступна тільки для читання
DateLastModified	Містить дату та час останньої зміни. Доступна тільки для читання
Drive	Містить необхідну букву пристрою, на якому знаходиться файл. Доступна тільки для читання
Name	Дозволяє переглянути і змінити ім'я файлу.
ParentFolder	Містить об'єкт ***Folder*** для батьківського каталогу файлу. Доступна тільки для читання
Path	Містить шлях до файлу
ShortName	Містить коротку назву файлу
ShortPath	Містить шлях до файлу, що складається з коротких імен каталогів
Size	Повертає розмір файлу в байтах
Type	Повертає інформацію про тип файлу

Наступний приклад дозволяє отримати розмір файлу запущеного скрипта і встановити його атрибут «**Тільки для читання**».

```
FSO = WScript.CreateObject («Scripting.FileSystemObject»);
file1 = FSO.GetFile (WScript.ScriptFullName);
WScript.Echo («Розмір файлу скрипта» + file1.Name + «:» + file1.Size);
file1.Attributes = 1;
```

Властивість ***Attributes*** є цілим числом, яке описує атрибути файлу згідно з табл. 7. Налаштування або читання окремого атрибута виконуються з використанням двійкової маски і порозрядних логічних операцій. У наведеному прикладі молодший біт властивості встановлюється в ***1***, що відповідає установці атрибуту «**Тільки для читання**».

Таблиця 7

Атрибути файлу у властивості Attributes об'єкта File

Атрибут	Значення	Опис
Normal	***0***	Звичайний файл без встановлених атрибутів
Readonly	***1***	Файл з атрибутом «лише для читання» (1-й біт)
Hidden	***2***	Прихований файл (2-й біт)
System	***4***	Системний файл (3-й біт)

Закінчення таблиці

Directory	***16***	Папка або каталог (можливо, з атрибутом «лише для читання») (5-й біт)
Archive	***32***	Файл з атрибутом «архівний» (6-й біт)
Alias	***1024***	Ярлик (.Lnk-файл) (11-й біт)
Compressed	***2048***	Стиснутий файл (12-й біт)

Для роботи з вмістом файла використовується об'єкт ***TextStream*** і його методи.

Типовими операціями є читання і запис інформації з/у файл.

Відкриття файлу (і створення об'єкта типу ***TextStream***) може здійснюватися методами ***CreateTextFile***, ***OpenTextFile*** і ***OpenAsText Stream***. Метод ***OpenAsTextStream*** належить об'єкту ***File***, він відкриває вже існуючий файл, для якого раніше був отриманий ***File*** (методом ***GetFile***). Методи ***CreateTextFile*** і ***OpenTextFile*** визначені в ***FSO*** і ***Folder***, вони можуть відкрити файл за іменем, причому, якщо файл, що відкривається не існує, можуть його створити. Обидва об'єкти повертають посилання на об'єкт типу ***TextStream***.

Синтаксис виклику методу ***OpenTextFile***:

FSO.OpenTextFile(filename[,iomode[,create[,format]]]), де

- ***filename*** – ім'я файлу;
- ***iomode*** – режим відкриття файлу, який може приймати значення;

– ***1*** – файл відкривається для читання, запис заборонений;

– ***2*** – файл відкривається для запису; якщо файл існує, його вміст видаляється;

– ***8*** – файл відкривається для додавання даних;

– ***create*** – логічна величина, що вимагає (при значенні **true**) створювати новий файл, якщо файлу з вказаним ім'ям не існує.

– ***format*** – кодування для відкриття файлу, може приймати значення;

– ***-2*** – відкривається з використанням системної кодування;

– ***-1*** – відкривається в кодуванні Unicode;

– ***0*** – відкривається в кодуванні ASCII.

```
var fso = WScript.CreateObject(«Scripting.FileSystemObject»)
var ts = fso.OpenTextFile(«d:/testfile.txt», 1, True)
```

Властивості і методи об'єкта TextStream наведені у таблиці 8.

Таблиця 8

Властивості і методи об'єкта TextStream

Властивість	**Опис**
Attributes	Дозволяє переглянути або встановити атрибути файлу
DateCreated	Містить дату та час створення файлу. Доступна тільки для читання
DateLastAccessed	Містить дату та час останнього звернення. Доступна тільки для читання
DateLastModified	Містить дату та час останньої зміни. Доступна тільки для читання
Drive	Містить необхідну букву пристрою, на якому знаходиться файл. Доступна тільки для читання
Name	Дозволяє переглянути і змінити ім'я файлу
ParentFolder	Містить об'єкт Folder для батьківського каталогу файлу. Доступна тільки для читання
Path	Містить шлях до файлу
ShortName	Містить коротку назву файлу
ShortPath	Містить шлях до файлу, що складається з коротких імен каталогів
Size	Повертає розмір файлу в байтах
Type	Повертає інформацію про тип файлу
Метод	**Опис**
Write(String)	Записує інформацію у файл в один рядок
WriteLine(String)	Записує інформацію, з переходом на новий рядок
WriteBlankLines()	Записує порожній рядок
Read(nChars)	Читає з файлу ***nChars*** символів
ReadLine()	Читає рядок з файлу
ReadAll()	Зчитує весь файл
Skip(nChars)	Пропускає ***nChars*** символів
SkipLine()	Пропускає рядок
Close()	Закриває відкритий файл

Розглянемо приклад, у якому для вирішення конкретного завдання використовуються перераховані методи і властивості об'єкта ***TextStream***. У наступному прикладі скрипт перебирає всі рядки файлу, ім'я якого передано йому в якості першого параметра, і виводить у вікно (разом з порядковим номером) тільки ті з них, у яких немає рядка «***rem***».

```
    var FSO = WScript.CreateObject(«Scripting.FileSystemObject»);
    if (WScript.Arguments.length>0 &&
FSO.FileExists(WScript.Arguments(0)))
    {
       var TextStream = FSO.OpenTextFile(WScript.Arguments(0));
       var n=1;
       while (!TextStream.AtEndOfStream)
       {
          Str = TextStream.ReadLine();
          if (Str.indexOf(«rem «)==-1)
             WScript.Echo(n+» «+Str);
     n++;
          }
          TextStream.Close();
       }
       else
          WScript.Echo(«Використання
скрипта:”+WScript.ScriptFullName + ”ім’я_файлу»);
```

Для роботи з папками і їх властивостями ***FSO*** надає об’єкт ***Folder*** і колекцію ***Folders***. ***Folder*** дозволяє працювати із заданою папкою. Багато його методів і властивостей схожі за призначенням з однойменними в об’єкті ***File***: методи ***Copy***, ***Move***, ***Delete***, властивості ***DateCreated***, ***DateLastAccessed***, ***DateLastModified***, ***Size***, ***Name***, ***Path***, ***Parent*** і т. ін. Відповідно, принципи роботи з папками аналогічні роботі з файлами (наприклад, поряд з методом ***FileExist FSO*** використовують метод ***FolderExist***, а поряд з методом ***GetFile*** можна використовувати метод ***GetFolder***). З властивостей, характерних для об’єкта ***Folder*** необхідно виділити колекції ***Files*** і ***SubFolders***, що містять списки файлів і папок заданої папки. З їх допомогою можна перебирати вміст папки.

Наступний приклад дозволяє вивести у вікно список програм, що автоматично запускаються, з меню «**Автозавантаження**»:

```
var WSHShell = WScript.CreateObject(«WScript.Shell»);
var fso = WScript.CreateObject(«Scripting.FileSystemObject»);
var StartFolder=fso.GetFolder(WSHShell.SpecialFolders(«Startup»));
var enFiles=new Enumerator(StartFolder.Files);
WScript.Echo («Список автозавантаження»);
for(; !enFiles.atEnd(); enFiles.moveNext())
{
  var cFile=enFiles.item();
  WScript.Echo( cFile.Name);
}
```

Наступний приклад демонструє, як можна рекурсивно обійти всі папки заданої папки (3-й параметр) і знайти в них файли із заданим розширенням (1-й параметр скрипта), які змінювалися в останній раз у заданий місяць (2-й параметр).

```
FSO = WScript.CreateObject(«Scripting.FileSystemObject»);

// рекурсивна функція перебору вмісту папки
function findFiles(cFolder, cExt, cDateMonth)
{
    var newFolder=true;

     // перебираємо всі файли чергової папки
     var cFiles=new Enumerator(cFolder.Files);
     for(;!cFiles.atEnd(); cFiles.moveNext())
     {
         var cFile=cFiles.item();

         // Створюємо об’єкт типу date с датою останньої
модифікації файлу
         var dateFile=new Date(cFile.DateLastModified);
         if(FSO.GetExtensionName(cFile.Name)==cExt &&
        dateFile.getMonth()==cDateMonth)
       {
         if(newFolder)
          {
            newFolder=false;
            WScript.Echo(«---- Папка: «+cFolder.Path+» -------»);
          }
          WScript.Echo(cFile.Name);
       }

    }

    // перебираємо всі підпапки і для кожної рекурсивно
викликаємо findFiles
    var cSubFlds=new Enumerator(cFolder.SubFolders);
    for(;!cSubFlds.atEnd(); cSubFlds.moveNext())
    {
      var cSFld=cSubFlds.item();
      findFiles(cSFld, cExt, cDateMonth);
    }
     }
//точка входу у скрипт
```

```
if(WScript.Argiments.length==3)
{
   var FileExt=WScript.Arguments(0);
   var FileMonth=WScript.Arguments(1);
   var workFolder=FSO.GetFolder(WScript.Arguments(2));
   findFiles(workFolder,FileExt,FileMonth);
}
else
{
   WScript.Echo("Использование скрипта:" + WScript.ScriptFull
Name + " расширение месяц папка");
}
```

Завдання

1. Для всіх файлів заданої папки, що мають розширення txt, встановити атрибути «**Тільки для читання**» і «**Прихований**».
2. Напишіть скрипт, який під час кожного запуску буде дописувати в текстовий файл інформацію про поточну дату і час, а також розмір папки, у якій розташований скрипт і кількість файлів у ній.
3. Написати скрипт, який би розбирав вміст папки, що задається як параметр запуску, і створював би папки відповідно до років створення файлів (2012, 2011 і т. ін.), всередині кожної з них створити папки з назвами місяців (січень , лютий, ..., грудень). Кожен файл зазначеної папки перенести у відповідну йому створену на поточний рік і місяць папку.

Контрольні питання

1. Які дії можна виконувати за допомогою об'єкта FileSystemObject у WSH?
2. Які об'єкти-колекції використовуються для роботи з файловою системою у WSH?
3. Як у сценарії WSH отримати інформацію про заданий каталог або файл?
4. Як у сценарії WSH перевірити існування певного диску, файлу або каталогу?
5. Як у сценарії WSH скопіювати файли або каталоги?
6. Як у сценарії WSH перемістити файли або каталоги?
7. Як у сценарії WSH відкрити текстовий файл для читання, запису або додавання?

Лабораторна робота № 7. Створення ярликів за допомогою сценаріїв WSH. Об'єкт Drive для отримання інформації про диск

Ще один корисний метод об'єкта ***WshShell*** – ***CreateShortcut***. З назви методу очевидно, що він дозволяє створити новий ярлик для деякого ресурсу. Параметр методу задає ім'я і повний шлях до створюваного ярлика. Створення ярлика здійснюється в три етапи:

1. Власне створення ярлика методом ***CreateSortcut.***
2. Встановлення властивостей ярлика. Перелік властивостей об'єкта ***WshShortcut*** наведений у таблиці 9.
3. Збереження ярлика з встановленими властивостями.

Таблиця 9

Властивості об'єкта WshShortcut

Властивість	Опис
TargetPath	Повне ім'я ресурсу, на який посилається ярлик
WindowStyle	Стиль вікна, що запускається. Задає вид вікна для ресурсу, що запускається
HotKey	Визначає комбінацію швидкого виклику ярлика
IconLocation	Розташування іконки ярлика, файли
Description	Підказка для ярлика
WorkingDirectory	Встановлює робочий каталог, який буде використовуватися застосунком для зберігання тимчасових і інших файлів

Розглянемо приклад створення нового ярлика для **Блокнота** в меню автозавантаження поточного користувача.

```
var WSHShell = WScript.CreateObject(«WScript.Shell»);
var StartPath = WSHShell.SpecialFolders(«Startup»);

//Створюємо ярлик в спецпапці Startup
var MyShortcut = WSHShell.CreateShortcut(StartPath + «\\ Блокнот .lnk»);

//Задаємо властивості для ярлика:
//Файл, що запускається (метод ExpandEnvironmentStrings розкриває системну змінну %windir% у її значення)
MyShortcut.TargetPath = WSHShell.ExpandEnvironmentStrings(«%windir%\\notepad.exe»);
```

```
    //Директорія файлу, що запускається
    MyShortcut.WorkingDirectory =
WSHShell.ExpandEnvironmentStrings(«%windir%»);

    //Тип вікна файлу, що запускається (запускаємо у згорнутому
вигляді)
    MyShortcut.WindowStyle = 7;

    //Іконка, що використовується для ярлика
    MyShortcut.IconLocation =
WSHShell.ExpandEnvironmentStrings(«%windir%\\notepad.exe, 0»);

    //Зберігаємо зміни властивостей ярлика
     MyShortcut.Save();
```

Для перебору всіх дисків комп'ютера (у тому числі змінних і мережевих) можна використовувати колекцію ***Drives***, кожен з елементів якої, будучи екземпляром об'єкта ***Drive***, описує один з дисків. Властивості ***Drive*** представлені у таблиці 10 (у цього об'єкта відсутні методи).

Таблиця 10

Властивості об'єкта Drive

Властивість	Опис
AvailableSpace	Обсяг доступного для користувача місця (в байтах) на диску
DriveLetter	Літера, асоційована з локальним пристроєм або мережевим ресурсом. Доступна тільки для читання
DriveType	Містить числове значення, що визначає тип пристрою: – ***0*** – невідомий пристрій; – ***1*** – пристрій зі змінним носієм; – ***2*** – жорсткий диск; – ***3*** – мережевий диск; – ***4*** – **CD-ROM**; (**CD-R** і **CD-RW** не розрізняються) – ***5*** – RAM-диск
FileSystem	Тип файлової системи, що використовується на диску (**FAT**, **NTFS** або **CDFS**)
FreeSpace	Обсяг вільного місця (в байтах) на локальному диску або мережевому ресурсі. Доступна тільки для читання
IsReady	Містить ***true***, якщо пристрій готовий, і ***false*** – у протилежному випадку.
Path	Містить шлях до диска
RootFolder	***Folder***, відповідний кореневому каталогу на диску. Доступна тільки для читання
SerialNumber	Десятковий серійний номер заданого диска

Закінчення таблиці

ShareName	Мережеве ім'я для диска. Якщо об'єкт не є мережевим диском, то у властивості ***ShareName*** міститься порожній рядок (*««*)
TotalSize	Загальний обсяг у байтах локального диска або мережевого ресурсу
VolumeName	Мітка тому для диска. Доступна для читання і запису

У наведеному нижче прикладі виводиться статистика щодо всіх жорстких дисків комп'ютера (літера диска, мітка тому, загальний розмір диска і обсяг вільного місця на диску). Одночасно підраховується сумарний обсяг і обсяг вільного місця для всіх дисків.

```
fso = WScript.CreateObject(«Scripting.FileSystemObject»);
var totalSize = 0, totalAvail = 0;
var cDrives = new Enumerator(fso.Drives);
for (;! cDrives.atEnd(); cDrives.moveNext())
{
  oDrive = cDrives.item();
  if (oDrive.DriveType == 2)
  {
      TotalSize + = oDrive.TotalSize;
    totalAvail + = oDrive.AvailableSpace;
   WScript.Echo (oDrive.DriveLetter + «(Volume:» +
oDrive.VolumeName +
    «): Total:» + Math.round (oDrive.TotalSize / 1024/1024) + «Mb,
avail:» +
     Math.round (oDrive.AvailableSpace / 1024/1024) + «Mb»);
  }
}
WScript.Echo («Загальний обсяг:» + Math.round (totalSize /
1024/1024) +
«Mb, вільний обсяг:» + Math.round (totalAvail / 1024/1024) +
«Mb»);
```

Завдання

1. Написати скрипт, який перевіряв би відповідність усіх ярликів на робочому столі реальних файлів на диску і знищував би ті ярлики, для яких файлу, на який він посилається, не існує. Список віддалених ярликів заносити в файл у форматі «Дата видалення: Ім'я ярлика, Шлях до пов'язаного файлу».

2. Написати скрипт, який приймає введену від користувача літеру певного диску та виводить інформацію щодо загального обсягу відповідного диску, а також про вільний його обсяг.

Контрольні питання

1. Для чого призначений метод CreateShortcut об’єкта WshShell?
2. Перелічіть основні властивості об’єкта WshShell та їх призначення.
3. Як у сценарії WSH отримати інформацію про певний диск?
4. Перелічіть основні властивості об’єкта Drive та їх призначення.

Лабораторна робота № 8. Запуск програмних застосувань зі сценаріїв WSH. Імітація натиснення клавіш у програмних застосуваннях

Метод ***Run*** об'єкта ***WshShell*** дозволяє запускати інші програми. Для другого параметра (***intWindowStyle***), що визначає зовнішний вигляд вікна програми яку ви запускаєте, допустимі значення, наведені у таблиці 11.

Таблиця 11

Можливі значення параметра intWindowStyle методу Run

Значення	Опис
0	Запуск у прихованому вигляді
1	Звичайний розмір вікна, якщо воно обернене або розгорнуте на весь екран, то йому повертається вихідний вигляд і положення на екрані
2	Запуск у згорнутому вигляді
3	Вікно розгортається на весь екран
4	Запуск у звичайному розмірі, в неактивному стані (без фокусу)
8	Звичайний розмір в неактивному стані, але у фокусі залишається запущене застосування
10	Вид вікна визначається додатком, який його викликав

Запустимо за допомогою методу ***Run*** в браузері **Internet Explorer** заданий сайт:

```
var wh = WScript.CreateObject («WScript.Shell»);
wh.Run («\»C:\\Program Files\\Internet Explorer\\iexplore.exe\ «www.ukr.net»);
```

Наведений приклад демонструє додатково аргумент методу ***Run:*** якщо в шляху присутні пробіли, шлях необхідно укладати в подвійні лапки. Під час використання **JScript** варто пам'ятати про екранування лапками спецсимволів (\ «і \\ в останньому прикладі).

За допомогою методу ***Run*** можна виконати консольну команду:

```
var wh=WScript.CreateObject(«WScript.Shell»);
var status=wh.Run("ping 192.168.0.1", 0, true);
if( status!=0) WScript.Echo("Не можу зв'язатися з віддаленим комп'ютером ");
```

В останньому прикладі метод ***Run*** виконує команду в прихованому вікні командного інтерпретатора, припиняючи роботу скрипту і повертаючи код завершення виконаної команди. Ще однією можливістю запустити консольний додаток зі скрипту є використання методу ***WShell.Exec***. На відміну від метода ***Run***, він створює новий процес як дочірній, надаючи батьківському скрипту можливості доступу до своїх об'єктів ***StdIn***, ***StdOut*** і ***StdErr*** і тим самим спрощуючи процес обміну інформацією. У наступному прикладі команда ***Exec*** не тільки запускає дочірній процес, але й створює об'єкт ***wse***, за допомогою властивості ***StdOut*** якого скрипт, який запускає, може отримати виведення скрипту, який запускається.

```
var wh=WScript.CreateObject(«WScript.Shell»);
var wse=wh.Exec(«netstat»);
str=wse.StdOut.ReadAll();
StrMas=str.split(«\n»);
for (i=4;i<StrMas.length;i++)
 {
  if (StrMas[i].indexOf(«:http»)>0)
WScript.Echo(StrMas[i].substring(StrMas[i].indexOf(«:»)+5,
StrMas[i].lastIndexOf(«:http»)));
 }
```

Наведений вище скрипт запускає команду ***netstat*** у дочірньому процесі і вибирає для його виведення інформацію тільки про сполуки за протоколом **http**.

Для взаємодії з запущеним додатком крім його власних інтерфейсних методів і властивостей можна використовувати стандартний метод класу ***WshShell SendKeys***. Він дозволяє імітувати клавіатурне введення для програми, що має фокус введення (який, в свою чергу, можна привласнити додатком методом ***AppActivate***). Наступний приклад відкриває стандартний **Блокнот** і заносить в нього рядок ***Hello, world!*** і зберігає текст на диску під ім'ям ***WSH.txt***.

```
var wh=WScript.CreateObject(«WScript.Shell»);
var wse=wh.Exec(«%windir%\\notepad.exe»);
WScript.Sleep(1000);
wh.AppActivate(wse.ProcessID);
var str=new String(«Hello world.»);
wh.SendKeys(string=str);
WScript.Sleep(1500);
wh.SendKeys(«{BACKSPACE}»);
wh.SendKeys(«!»);
```

```
WScript.Sleep(1500);
var Answer=wh.Popup(«Зберегти документ? \n(Інакше закрити блокнот)»,10,»WSH»,36);
wh.AppActivate(wse.ProcessID);
WScript.Sleep(1500);
if (Answer==6)
{
    wh.SendKeys(«^s»);
    WScript.Sleep(100);
    wh.SendKeys(«WSH.txt»);
    wh.SendKeys(«{ENTER}»);
}
else
   wse.Terminate();
for (;;)
{
   if (wse.Status==1) break;
   WScript.Sleep(100);
}
WScript.Echo(«Блокнот завершений с кодом «+wse.ExitCode);
```

В останньому прикладі звертає на себе увагу спосіб відправки повідомлень про натискання клавіші ***S*** в комбінації з ***Ctrl*** (***^S***). Аналогічно комбінація з ***Alt*** відправляється з символом ***%*** (наприклад, ***%A***), а комбінація з ***Shift*** – з символом **+** (наприклад, ***+D***). Для спецклавіш, таких як функціональні, ***Enter***, ***Delete*** і інших використовуються рядкові позначення, які необхідно укладати в фігурні дужки. Наприклад, закрити активне вікно можна командою ***wh.SendKeys(«%{F4}»)***. Більш докладно про функції ***SendKeys*** (методи емуляції комбінацій клавіш, автоповтору натискань, строкових позначеннях спецклавіш і ін.) можна подивитися в документації.

Ще одна деталь останнього скрипту, що не зустрічалася раніше – використання стандартного діалогового вікна викликом методу ***WScript.Popup***. Він дозволяє визначати текст повідомлення і заголовка вікна, набір кнопок у вікні. Як результат повертається код натиснутої кнопки.

Завдання

1. Написати скрипт, який запускає редактор реєстру, або будь-який інший додаток.
2. Написати скрипт, який запускає стандартний калькулятор і обчислює у ньому значення виразу (12345-678)/910.

Контрольні питання

1. Як з коду скрипту виконати сторонній скрипт або запустити застосування? Які засоби дозволяють керувати стороннім застосуванням?
2. Як у скрипті WSH імітувати натиснення певних клавіш у запущених програмних застосуваннях?
3. Яким чином в ході імітації натиснення клавіш задаються звичайні та спеціальні клавіші? Наведіть приклади.

Лабораторна робота № 9. Введення до оболонки Windows PowerShell. Основи роботи у командному рядку PowerShell

Нова оболонка **Windows PowerShell** була задумана розробниками **Microsoft** як більш потужне середовище для написання сценаріїв та роботи у командному рядку. Розробники **PowerShell** переслідували кілька цілей, головна з яких – створення середовища складання сценаріїв, яке найкращим чином підходило б для сучасних версій **ОС Windows** і було б більш функціональним, розширюваним простим у використанні, ніж будь-який аналогічний продукт для будь-якої іншої ОС. Насамперед це середовище повинно було підходити для вирішення завдань, що стоять перед системними адміністраторами, а також задовольняти вимогам розробників програмного забезпечення, надаючи їм засоби для швидкої реалізації інтерфейсів управління до створюваних додатків.

Запуск оболонки. Виконання команд

Для запуску оболонки варто натиснути на кнопку **Пуск (Start)**, відкрити меню **Всі програми (All Programs)**, вибрати елемент **Стандартні**, **Windows PowerShell** і **Windows PowerShell ISE**. Інший варіант запуску оболонки – пункт **Виконати ... (Run)** в меню **Пуск (Start)**, ввести ім'я файлу **powershell_ise** і натиснути кнопку **OK**.

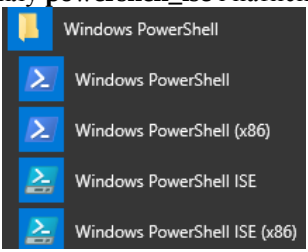

Рис. 1. Запуск **PowerShell ISE** за допомогою меню

У результаті відкриється нове командне вікно із запрошенням вводити команди (рис. 2).

У нижній частині вікна вводяться команди. Середня частина вікна містить результати виконання введеної команди або повідомлення про помилки.

Верхня частина використовується для роботи з командними файлами.

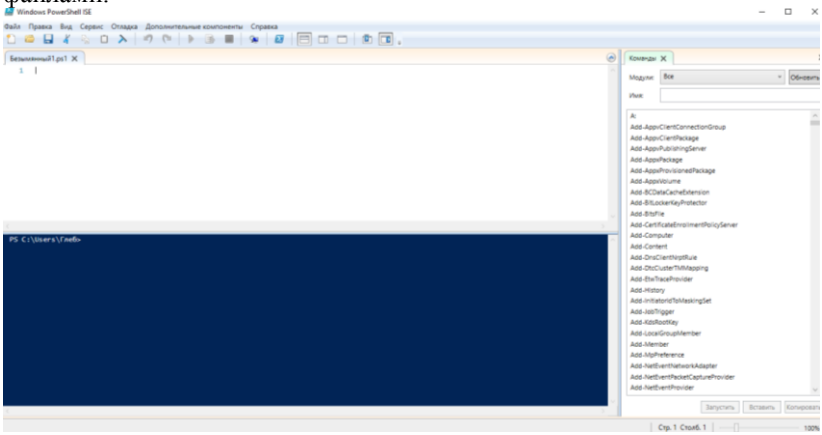

Рис. 2. Командне вікно оболонки **PowerShell ISE**

Типи команд PowerShell

В оболонці **PowerShell** підтримуються команди чотирьох типів: командлети, функції, сценарії і зовнішні виконувані файли.

Перший тип – так звані **командлети (cmdlet)**. Цей термін використовується поки тільки всередині **PowerShell**. Командлет – аналог внутрішньої команди інтерпретатора командного рядка – являє собою клас **.NET**, породжений від базового класу ***Cmdlet***; розробляються командлети за допомогою пакета **PowerShell Software Developers Kit (SDK)**. Єдиний базовий клас ***Cmdlet*** гарантує сумісний синтаксис всіх командлетів, а також автоматизує аналіз параметрів командного рядка і опис синтаксису командлетів для вбудованої довідки.

Вищезазначені команди компілюються в динамічну бібліотеку (DLL) і підвантажуються до процесу **PowerShell** під час запуску оболонки (тобто самі собою командлети не можуть бути запущені як додатки, але в них містяться виконувані об’єкти). Командлети – це аналог внутрішніх команд традіційних оболонок.

Наступний тип команд – **функції**. Функція – це блок коду на мові **PowerShell**, що має назву і знаходиться в пам’яті до завершення поточного сеансу командної оболонки. Функції, як і командлети, підтримують іменовані параметри. Аналіз синтаксису функції проводитися один раз під час її оголошення.

Сценарій – це блок коду на мові **PowerShell**, що зберігається в зовнішньому файлі з розширенням ***ps1***. Аналіз синтаксису сценарію проводитися в ході кожного його запуску.

Останній тип команд – **зовнішні виконувані файли**, які виконуються звичайним чином операційною системою.

Імена і синтаксис командлетів

У **PowerShell** аналогом внутрішніх команд є командлети. Командлети можуть бути дуже простими або дуже складними, але кожен з них розробляється для вирішення однієї, вузької задачі. Робота з командлетами стає по-справжньому ефективною під час використання їх композицій (конвеєризації об'єктів між командлетами).

Команди **Windows PowerShell** відповідають певним правилам іменування:

– команди складаються з дієслова та іменника (завжди в однині), розділених тире. Дієслово задає певну дію, а іменник визначає об'єкт, над яким цю дію буде здійснено. Команди записуються англійською мовою. Приклад: ***Get-Help*** викликає інтерактивну довідку щодо синтаксису **Windows PowerShell**;

– перед параметрами ставиться символ «-». Наприклад: ***Get-Help -Detailed***.;

– у **Windows PowerShell** також включені псевдоніми багатьох відомих команд. Це спрощує знайомство і використання **Windows PowerShell**. Приклад: команди ***help*** (класичний стиль **Windows**) і ***man*** (класичний стиль **UNIX**) працюють так само, як і ***Get-Help***.

Наприклад, ***Get-Process*** (отримати інформацію про процес), ***Stop-Service*** (зупинити службу), ***Clear-Host*** (очистити екран консолі) і т. ін. Щоб переглянути список командлетів, доступних в ході поточного сеансу, потрібно виконати командлет ***Get-Command***.

За замовчуванням командлет ***Get-Command*** інформує вас в трьох стовпцях: ***CommandType***, ***Name*** і ***Definition***. У цьому разі в стовпці ***Definition*** відображається синтаксис командлетів (три крапки (**...**) у стовпці синтаксис вказує на те, що дані обрізані).

Зауваження. Косі риски (/ і \) разом з параметрами в оболонці **Windows PowerShell** не використовуються.

У загальному випадку синтаксис командлетів має наступну структуру:

імя_командлета -параметр1 -параметр2 аргумент1 аргумент2.

Тут ***параметр1*** – параметр (перемикач), що не має значення; ***параметр2*** – ім'я параметра, що має значення ***аргумент1***; ***аргумент2*** – параметр, який не має імені. Наприклад, командлет ***Get-Process*** має параметр ***Name***, який визначає ім'я процесу, інформацію про який потрібно вивести. Ім'я цього параметра вказувати необов'язково. Таким чином, для отримання відомостей про процес **Far** можна ввести або команду ***Get-Process -Name Far***, або команду ***Get-Process Far***.

Псевдоніми команд

Механізм псевдонімів, реалізований в оболонці **PowerShell**, дає можливість користувачам виконувати команди за їхніми альтернативними іменами (наприклад, замість команди ***Get-Childitem*** можна користуватися псевдонімом ***dir***). У **PowerShell** заздалегідь визначено багато псевдонімів, можна також додавати власні псевдоніми в систему.

Псевдоніми в **PowerShell** діляться на два типи. Перший – призначений для сумісності імен з різними інтерфейсами. Псевдоніми цього типу дозволяють користувачам, які мають досвід роботи з іншими оболонками (**Cmd.exe** або **UNIX-оболонки**), використовувати знайомі їм імена команд для виконання аналогічних операцій в **PowerShell**, що спрощує освоєння нової оболонки, дозволяючи не витрачати зусиль на запам'ятовування нових команд **PowerShell**. Наприклад, користувач хоче очистити екран. Якщо у нього є досвід роботи з **Cmd.exe**, то він, природно, спробує виконати команду ***cls***. **PowerShell** при цьому виконає командлет ***Clear-Host***, для якого ***cls*** є псевдонімом і який виконує потрібні дії – очищення екрану. Для користувачів **Cmd.exe** в **PowerShell** визначені псевдоніми ***cd, cls, copy, del, dir, echo, erase, move, popd, pushd, ren, rmdir, sort, type***; для користувачів **UNIX** – псевдоніми ***cat, chdir, clear, diff, h, history, kill, lp, ls, mount, ps, pwd, r, rm, sleep, tee, write***.

Дізнатися, який саме командлет ховається за знайомим псевдонімом, можна за допомогою командлета ***Get-Alias***:

```
PS C:\> Get-Alias cd

CommandType     Name        Definition
-----------     ----        ----------
Alias           cd          Set-Location
```

Псевдоніми другого типу (стандартні псевдоніми) в **PowerShell** призначені для швидкого введення команд. Такі псевдоніми утворюються з імен командлетів, яким вони відповідають. Наприклад, дієслово ***Get*** скорочується до ***g***, дієслово ***Set*** – до ***s***, іменник ***Location*** –

до ***l*** і т. ін. Таким чином, командлету ***Set-Location*** відповідає псевдонім ***sl***, а ***Get-Location*** – псевдонім ***gl***.

Переглянути список усіх пседонімів, оголошених в системі, можна за допомогою командлета ***Get-Alias*** без параметрів. Визначити власний псевдонім можна за допомогою командлета ***Set-Alias***.

Довідкова система PowerShell

У **PowerShell** передбачено кілька способів отримання довідкової інформації всередині оболонки.

Коротку довідку щодо одного командлета можна отримати за допомогою параметра ***?*** (знак питання), зазначеного після імені цього командлета. наприклад:

```
PS C:\> Get-Process -?
```

Для отримання докладної інформації про команд лети, використовують спеціальний командлет ***Get-Help***, який варто запускати з параметрами ***-Detailed*** або ***-Full***. Ключ ***-Full*** призводить до виведення всієї наявної довідкової інформації, а під час використання ключа ***-Detailed*** деяка технічна інформація опускається. В обох випадках будуть виведені докладні описи кожного з параметрів, підтримуваних цим командлетом.

Комана ***Get-Help*** дозволяє переглядати довідкову інформацію не тільки про різні командлети, але й синтаксис мови **PowerShell**, псевдоніми і тощо.

Наприклад, щоб прочитати докладну інформацію щодо використання масивів у **PowerShell**, потрібно виконати наступну команду: ***Get-Help about_array***.

Командлет ***Get-Help*** виводить вміст розділу довідки на екран відразу повністю. Функції ***man*** і ***help*** дозволяють довідкову інформацію виводити поекранно (аналогічно команді ***more*** інтерпретатора **Cmd.exe**), наприклад: ***man about_array***.

Навігація в файловій системі

В оболонці **cmd.exe** зміна поточного каталогу проводиться за допомогою команди ***cd***. У **PowerShell** команда ***cd*** має таке саме значення, вона також є стандартним псевдонімом командлета ***Set-Location***. Наприклад, наступна команда робить поточним каталог ***C:\Windows:***

```
PS C:\> cd C:\Windows:
PS C:\Windows>
```

Як і в оболонці **cmd.exe** як шлях можна вказувати символи **..** (для переходу в батьківський каталог) і \ (для переходу в кореневий каталог

поточного диску). Але потрібно враховувати й наступний нюанс. Людина, яка часто користувалася командою ***cd*** в оболонці ***cmd.exe***, швидше за все, буде машинально набирати команди типу ***cd..*** або ***cd*** без додаткових пробілів. У **PowerShell** це викличе помилку.

Ця помилка пов'язана з тим. що в **PowerShell** параметри команди завжди повинні відділятися від імені самої команди пробілом. Тому останню команду потрібно виконувати наступним чином:

```
PS C:\Windows> cd \
```

Отримання списку файлів і каталогів

Нагадаємо, що в оболонці **cmd.exe** список файлів і каталогів формується за допомогою внутрішньої команди ***dir***. У **PowerShell** також можна використовувати команду ***dir***, яка є псевдонімом командлета ***Get-ChildItem***.

У шляху, який вказується для команди ***dir***, можна застосовувати групові символи підстановки. Наприклад, наступна команда виведе всі файли з розширенням ***log*** з каталогу ***C:\Windows***:

```
PS C:\> dir C:\Windows\*.log
```

Параметр ***-Exclude*** дозволяє задати маску файлів, які не будуть оброблюватися командою ***dir***. Наприклад, наступна команда виведе всі файли з розширенням ***log*** з каталогу ***C:\Windows***, крім тих, чиє ім'я починається на букву ***d***:

```
PS C:\> dir C:\Windows \*.log -Exclude d*.log
```

Параметр ***-Name*** дозволяє виводити на екран тільки імена файлів (таким чином, цей параметр є аналогом ключа ***/b*** команди ***dir*** з **cmd.exe**), наприклад:

```
PS C:\> dir C:\Windows \*.log –Name
```

Параметр ***-Recurse*** вмикає режим рекурсії, під час якого командлет ***dir*** відображає не тільки вміст зазначеного каталогу, але й всіх його підкаталогів:

```
PS C:\> dir 'Documents and Settings' –Recurse.
```

За замовчуванням командлет ***dir*** не «бачить» приховані файли. Якщо їх необхідно також включати в список, то потрібно вказати параметр ***-Force*****:**

```
PS C:\> dir –Force.
```

Створення файлів і каталогів

Створити новий файл або каталог у **PowerShell** дозволяє командлет ***New-Item***. Шлях до створюваного елементу вказується у вигляді

значення параметра ***-Path***, а в якості значення параметра ***-Type*** вказується «***directory***», якщо потрібно створити каталог, і «***file***», якщо потрібно створити файл. Наприклад, наступна команда створює на диску ***C:*** каталог з ім'ям ***test folder***:

PS C:\> New-Item -Path C:\test_folder -Type «directory»

Під час створення файлу в нього відразу можна записати рядок, вказавши його як значення параметра ***-Value***, наприклад:

PS C:\> New-Item -Path C:\test_file.txt -Type «file» -Value «Test»

Якщо спробувати назвати створюваний файл ім'ям вже існуючого файлу:

PS C:\> New-Item -Path C:\test_file.txt -Type «file» -Value «Test»

то виникне помилка.

Для перезапису існуючого файлу у процесі створення потрібно вказати параметр ***-Force***:

PS C:\> New-Item -Path C:\test_file.txt -Type «file» -Value «Test2» –Force

Читання і перегляд вмісту файлів

В оболонці **cmd.exe** є команда ***type***, яка виводить вміст текстового файлу на екран. У **PowerShell** команда ***type*** є псевдонімом командлету ***Get-Content*** (інші псевдоніми цього ж командлету – ***cat*** і ***gc***), призначеного для порядкового зчитування вмісту текстового файлу з поверненням об'єкта для кожного рядка (рядки відображаються на екрані). Наприклад:

PS C:\> Get-Content C:\Windows\win.ini

Параметр ***-Encoding*** командлет ***Get-Content*** дозволяє дійсно вказувати кодування файлу для коректної обробки його вмісту. Допустимі значення цього параметра: ***Unicode***, ***Byte***, ***BigEndian Unicode***, ***UTF8***, ***UTF7***, ***Ascii***.

За замовчуванням командлет ***Get-Content*** зчитує всі рядки з файлу; їх кількість можна обмежити за допомогою параметра ***-TotalCount***. Наприклад, наступна команда зчитує перші п'ять рядків з файлу ***C:\Windows\win.ini***:

PS C:\> Get-Content C:\Windows\win.ini -TotalCount 5

Запис файлів

Записати інформацію у зовнішні файли можна за допомогою операторів перенаправлення (> і >>) і командлета ***Out-File***. При цьому командлет ***Out-File*** намагатиметься форматувати об'єкти, що

записуються в файл. Якщо потрібно просто записати в файл текстову інформацію (без додаткового форматування), то краще скористатися командлетом ***Set-Content***.

Інформації для запису в файл можуть задаватися як значення параметра ***-Value***. Наприклад, наступна команда записує в файл ***C:\test.txt*** рядок ***«Рядок з PowerShell»***:

```
PS C:\> Set-Content C:\test.txt -Value «Рядок з PowerShell»
```

Копіювання файлів і каталогів

У **PowerShell** копіювання файлів і каталогів здійснюється командлетом ***Copy-Item***, який є псевдонімом ***copy***. Шлях до файлів, що копіюються, вказується у якості значення параметра ***-Path*** (цей параметр використовується за замовчуванням), а шлях до цільового каталогу, в який потрібно скопіювати файли, задається значенням параметра **–** ***Destination***. Наприклад, наступна команда скопіює файл ***styles.css*** з кореневого каталогу диска з: в каталог ***C:\test_folder***:

```
PS C:\> copy C:\Styles.css -Destination C:\test_folder
```

Для того щоб побачити результат виконання команди копіювання, потрібно вказати параметр ***-PassThru***:

```
PS C:\> copy C:\styles.css -Destination C:\test_folder –PassThru
```

Якщо шлях до об’єктів, що копіюються, вказує на каталог, то за замовчуванням буде скопійований тільки цей каталог без свого вмісту (цим **PowerShell** відрізняється від більшості інших оболонок, у тому числі від ***cmd.exe***). Наприклад:

```
PS C:\> copy C:\script -Destination C:\test_folder -PassThru
```

Параметр ***-Recurse*** дозволяє копіювати вміст вкладених каталогів, наприклад:

```
PS C:\> copy C:\script -Destination C:\test_folder -Recurse -PassThru
```

Можна копіювати не всі файли з каталогу, а тільки відповідні певній масці. Її можна вказати всередині шляху для копіювання або в якості значення параметра ***-Include***. Наприклад, наступна команда копіює всі файли з розширенням ***psl*** з каталогу ***C:\script*** в папку ***C:\test_folder***:

```
PS C:\> copy C:\script\*.psl -Destination C:\test_folder –PassThru
```

Однак якщо необхідно скопіювати і підкаталоги, то одним командлетом ***Copy-Item*** обійтися не вдасться. Попередньо необхідні файли потрібно отримати командлетом ***Get-ChildItem (dir***), а потім передати їх командлету ***Copy-Item*** конвеєром. Наприклад, наступна

команда копіює всі файли з розширенням ***psl*** з каталогу ***C:\script*** і всіх його підкаталогів у папку ***C:\test_folder***:

```
PS C:\> dir -Recurse -Include *.psl c:\script\* | copy -Destination C:\test_folder -PassThru
```

Команда ***copy*** оболонки **cmd.exe** дозволяла об'єднувати кілька файлів (конкатенація файлів). У **PowerShell** об'єднати файли можна за допомогою командлету ***Get-Content*** (псевдонім ***type***) і перенаправлення виведення в результуючий файл. Розглянемо приклад. Створимо файли ***l.txt*** і ***2.txt***:

```
PS C:\> New-Item -Path C:\l.txt -Type «file» -Value «File 1»
PS C:\> New-Item -Path C:\2.txt -Type «file» -Value «File 2»
```

Наступна команда об'єднує файли ***1.txt*** і ***2.txt*** у файл ***3.txt***:

```
PS C:\> type l.txt, 2.txt > .\3.txt
```

Перейменування і переміщення файлів і каталогів

Перейменувати файл або каталог можна за допомогою командлета ***Rename-Item*** (псевдонім ***ren***). Значення параметра ***-Path*** цього командлета задає шлях до елементів для перейменування, а значення параметра ***–NewName*** – нове ім'я. Імена параметрів можна опускати (у цьому випадку першим має зазначатися значення параметра ***-Path***). Наприклад, створимо файл ***c:\l.tmp*** і перейменуємо його у файл ***2.tmp***:

```
PS C:\> New-Item -Path C:\l.tmp -Type «file»
PS C:\> ren l.tmp 2.tmp
```

Для того щоб побачити результат дії командлету ***Rename-Item***, потрібно вказати параметр ***-PassThru***:

```
PS C:\> ren 2.tmp 3.tmp –PassThru
```

Командлет ***Rename-Item*** дозволяє лише перейменовувати файли або каталоги, а не переміщувати їх. Якщо потрібно перемістити файл або каталог в іншу папку, то треба скористатися командлетом ***Move-item*** (псевдонім ***move***). Значення параметра ***-Path*** командлету задає шлях до файлів або каталогів для переміщення (у цьому шляху допускається використання символів узагальнення), а значення параметра ***–Destination*** - шлях до каталогу. куди будуть переміщені файли або каталоги. Результат переміщення можна побачити на екрані, вказавши параметр ***-PassThru***. Наприклад, наступна команда перенесе у кореневий каталог диску ***C:*** каталог ***C:\test_folder\folderl*** з усім його вмістом:

```
PS C:\> Move-Item -Path C:\test_folder\folderl C:\ -PassThru
```

Видалення файлів і каталогів

Видаляти об'єкти файлової системи можна за допомогою командлета ***Remove-Item*** (псевдонім ***del***). Значення параметра ***-Path*** задає шлях до файлів або каталогів, що видаляються (ім'я параметра в команді можна не вказувати). У шляху допускаються групові символи. Крім того, командлет ***Remove-Item*** має параметр ***–Include***, значення якого задає файли, на які діятиме команда, і параметр ***-Exclude***, що задає файли-виключення, які видалятися не будуть.

Наприклад, наступна команда видалить всі файли з розширенням ***psl*** в каталозі ***C:\test_folder***:

```
PS C:\> del C:\test_folder\*.psl
```

Якщо спробувати видалити всі файли в каталозі, що має підкаталоги, то система видасть попередження:

```
PS C:\> del C:\test_folder\*
```

Завдання

Для виконання завдань використовувати командну оболонку Windows PowerShell!

1. Переглянути вміст поточної директорії.
2. Переглянути вміст директорії N:\Scripts (або будь-якої іншої на ваш розсуд), виводячи тільки імена файлів.
3. Створити в кореневому каталозі диску N каталог Work.
4. Скопіювати в каталог Work всі файли з розширенням .bat з ЛР № 4.
5. Створити в каталозі Work підкаталог Sub і скопіювати в нього файли, що починаються з літери «t» з обраного на ваш розсуд каталогу на диску N.
6. Створити в каталозі Sub копії bat-файлів, змінивши розширення на .txt, але залишивши ім'я.
7. Переглянути вміст каталогу Work.
8. Видалити файли з розширенням .bat з каталогу Work із запитом на підтвердження, а потім знову переглянути вміст каталогу Work.
9. Створити підкаталог Temp в каталозі Sub.
10. У каталозі Temp створити файл mail.txt з вашою електронною адресою і файл surname.txt з вашим прізвищем.
11. Об'єднати файли mail.txt і surname.txt в all.txt і переглянути його вміст на екрані.
12. Знищити створені каталоги і їх вміст після закінчення роботи.

Контрольні питання

1. У чому полягає основна різниця оболонки Windows PowerShell від командного рядка?
2. Для чого призначена утиліта Windows PowerShell ISE?
3. На які типи поділяються команди PowerShell?
4. Що таке командлет? Яку структуру він має?
5. Псевдоніми яких стандартних команд Windows та Linux існують у PowerShell? Які командлети насправді використовуються під час виклику цих команд?
6. Як вивести довідку про певну команду у PowerShell?

Лабораторна робота № 10. Управління виведенням команд у Windows PowerShell. Перегляд структури, фільтрація та сортування об'єктів

Конвеєризація і управління виведенням команд Windows PowerShell

Раніше було розглянуто поняття конвеєризації (або композиції) команд інтерпретатора **Cmd.exe**, коли вихідний потік однієї команди перенаправлявся у вхідний потік іншої, об'єднуючи тим самим дві команди разом. Подібні конвеєри команд використовуються в більшості оболонок командного рядка і є засобом, що дозволяє передавати інформацію між різними процесами. Механізм композиції команд представляє, ймовірно, найбільш цінну концепцію, яка використовується в інтерфейсах командного рядка. Конвеєри не тільки знижують зусилля, докладені під час введення складних команд, але й полегшують відстеження потоку роботи в командах.

В оболонці **PowerShell** також дуже широко використовується механізм конвеєризації команд, проте тут конвеєром передається не потік тексту, як у всіх інших оболонках, а **об'єкти**. У цьому разі з елементами конвеєра можна виробляти різні маніпуляції: фільтрувати об'єкти за певним критерієм, сортувати і групувати їх, змінювати структуру (нижче ми докладніше розглянемо операції фільтрації і сортування елементів конвеєра).

Конвеєризація об'єктів у PowerShell

Конвеєр в **PowerShell** – це послідовність команд, розділених між собою знаком | (Вертикальна риска). Кожна команда в конвеєрі отримує об'єкт від попередньої команди, виконує певні операції над ним та передає наступній команді в конвеєрі. З точки зору користувача, об'єкти упаковують пов'язану інформацію в форму, в якій інформацією простіше маніпулювати як єдиним блоком і з якої за необхідності витягуються певні елементи.

Передача інформації між командами у вигляді об'єктів має велику перевагу над звичайним обміном інформацією за допомогою потоку тексту. Адже команда, яка бере потік тексту від іншої утиліти, повинна його проаналізувати, розібрати і виділити потрібну їй інформацію, а це може бути непросто, оскільки зазвичай виведення команди більше орієнтоване на візуальне сприйняття людиною (це природно для

інтерактивного режиму роботи), а не на зручність подальшого синтаксичного розбору.

Під час передачі конвеєром об'єктів цієї проблеми не виникає, тут потрібна інформація витягується з елемента конвеєра простим зверненням до відповідної властивості об'єкта. Однак виникає нове запитання: яким чином дізнатися, які саме властивості є у об'єктів, що передаються конвеєром? Адже у процесі виконання того чи іншого командлету ми на екрані бачимо тільки одну або кілька колонок відформатованого тексту.

Приклад. Запустимо командлет ***Get-Process***, який виводить інформацію про запущені в системі процеси:

```
PS C:\> Get-Process
```

```
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
     80       4      876        418       1780           0 AsLdrSrv
   3372      44    61296      65988       1812           0 AvastSvc
    989      24    17316      36228       6,80        8124 3 AvastUI
    107       5     2592      10488       0,33        3184 3 conhost
-------------------------------------------------------------------------------
    746      21    65836      65372      63,73        2576 3 WINWORD
    117       4      900       5268                   2224 0 WmiApSrv
    260       6     1492       7456                   4956 0 WUDFHost
```

Фактично на екрані ми бачимо тільки зведену інформацію (результат форматування отриманих даних), а не повне представлення вихідного об'єкта. З цієї інформації незрозуміло, скільки точно властивостей є у об'єктів, що генеруються командою ***Get-Process***, і які імена мають ці властивості. Наприклад, ми хочемо знайти всі «завислі» процеси, які не відповідають на запити системи. Чи можна це зробити за допомогою командлета ***Get-Process***, яку властивість потрібно перевіряти у виведених об'єктів?

Для відповіді на ці запитання потрібно навчитися досліджувати структуру об'єктів **PowerShell**, дізнаватися, які властивості і методи вони мають.

Перегляд структури об'єктів

Для аналізу структури об'єкта, що повертається певною командою, найпростіше направити його конвеєром на командлет ***Get-Member*** (псевдонім ***gm***), наприклад:

```
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;}
----------------------------------------------------------------
```

Тут ми бачимо ім'я .NET-класу, екземпляри якого повертаються в ході роботи досліджуваного командлету (в нашому прикладі це клас ***System.Diagnostic.Process***), а також повний список елементів об'єкта (зокрема, цікавить нас властивість ***Responding***, що визначає «завислі» процеси). На екран також виводиться дуже багато елементів, переглядати їх незручно. Командлет ***Get-Member*** дозволяє перерахувати тільки ті елементи об'єкта, які є його властивостями. Для цього використовується параметр ***-MemberType*** зі значенням ***Properties***:

```
PS C:\> Get-Process | Get-Member -MemberType Property
TypeName: System.Diagnostics.Process

Name                MemberType      Definition
----                ----------      ----------
BasePriority        Property        System.Int32 BasePriority {get;}
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;}
```

Процесам ОС відповідають об'єкти, що мають дуже багато властивостей, у процесі роботи командлет ***Get-Process*** на екран виводить лише кілька з них (способи відображення об'єктів різних типів задаються файлами форматі XML, що знаходяться в каталозі, де встановлений файл **powershell.exe**).

Розглянемо найбільш часто використовувані операції над елементами конвеєра: фільтрації і сортування.

Фільтрація об'єктів у конвеєрі

У **PowerShell** підтримується можливість фільтрації об'єктів у конвеєрі, тобто видалення з нього об'єктів, які задовольняють певній умові. Таку функціональність забезпечує командлет ***Where-Object***, що дозволяє перевірити кожен об'єкт, що знаходиться в конвеєрі, і передати його далі конвеєром, тільки якщо він задовольняє умовам перевірки.

Наприклад, для виведення інформації про «завислі» процеси (об'єкти, які повертаються командлетом ***Get-Process***, у яких властивість ***Responding*** дорівнює ***False***) можна використовувати наступний конвеєр:

Get-Process | Where-Object {-not $_.Responding}.

Інший приклад – залишимо в конвеєрі тільки ті процеси, у яких значення ідентифікатора (властивість ***Id***) більше ***1000***:

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

У блоках сценаріїв командлет ***Where-Object*** для звернення до поточного об'єкту конвеєра і вилучення потрібних його властивостей використовується спеціальна змінна ***$_***, яка створюється оболонкою **PowerShell** автоматично. Таке змінна використовується і в інших командлетах, які виробляють обробку елементів конвеєра.

Умова перевірки в ***Where-Object*** задається у вигляді блоку сценарію – однієї або декількох команд **PowerShell**, укладених у фігурні дужки ***{}***. Результатом виконання зазначеного блоку сценарію має бути значення логічного типу: ***True*** або ***False***). Як можна зрозуміти з прикладів, у блоці сценарію використовуються спеціальні оператори порівняння.

Зауваження. У **PowerShell** для операторів порівняння не використовуються звичайні символи > або <, оскільки у командному рядку вони зазвичай означають перенаправлення введення/виведення.

Основні оператори порівняння наведені в табл. 12.

Таблиця 12

Оператори порівняння в PowerShell

Оператор	Значення	Приклад (повертається значення True)
-eq	дорівнює	***10 -eq 10***
-ne	не дорівнює	***9 -ne 10***
-lt	менше	***3 -lt 4***

Закінчення таблиці

-le	менше або дорівнює	***3 –le 4***
-gt	більше	***4 -gt 3***
-ge	більше або дорівнює	***4 -ge 3***
-like	порівняння на збіг з урахуванням символів узагальнення в тексті	***«file.doc» –like «f*.doc»***
-notlike	порівняння на розбіжність з урахуванням символів узагальнення в тексті	***«file.doc» –notlike «f*.rtf»***
-contains	містить	***1,2,3 –contains 1***
-notcontains	не містить	***1,2,3 –notcontains 4***

Оператори порівняння можна з'єднувати одне з одним за допомогою логічних операторів (табл. 13).

Таблиця 13

Логічні оператори в PowerShell

Оператор	Значення	Приклад (повертається значення True)
-and	логічне І	***(10 -eq 10) –and (1 –eq 1)***
-or	логічне АБО	***(9 -ne 10) –or (3 –eq 4)***
-not	логічне НЕ	***-not (3 –gt 4)***
!	логічне НЕ	***!(3 -gt 4)***

Сортування об'єктів

Сортування елементів конвеєра – ще одна операція, яка часто застосовується під час конвеєрної обробки об'єктів. Цю операцію здійснює командлет ***Sort-Object***: йому передаються імена властивостей, за якими потрібно провести сортування, а він повертає відомості, впорядковані за значеннями цих властивостей.

Наприклад, для виведення списку запущених в системі процесів, впорядкованим за витраченим процесорним часом (властивість ***cpu***), можна скористатися наступним конвеєром:

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

Для сортування в зворотному порядку використовується параметр ***Descending***:

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

У розглянутих нами прикладах конвеєри складалися з двох командлетів. Це не обов'язкова умова, він може об'єднувати і більшу кількість команд, наприклад:

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

У **PowerShell** є командлет ***Select-Object***, за допомогою якого можна виділяти вказану кількість об'єктів з початку або з кінця конвеєра, вибирати унікальні об'єкти з конвеєра, а також виділяти певні властивості в об'єктах, що проходять конвеєром.

Для виділення з конвеєра декількох перших або останніх об'єктів варто скористатися параметрами ***-First*** або ***–Last*** командлета ***Select-Object***. Наприклад, наступний конвеєр команд виведе на екран інформацію про п'ять процесів, що використовують найбільший обсяг пам'яті:

PS C:\> Get-Process | Sort-Object WS | Select-Object -Last 5

```
Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   1025      51    81192     136852   417    94,84   3360 chrome
    815     328   264024     154796   453           936 sqlservr
    274      45   135896     209136   932    25,06   5064 chrome
    328      40    94880     236864   962    52,09   6232 chrome
    749     628   414860     244072   974           2024 msmdsrv
```

Перший командлет у конвеєрі (***Get-Process***) повертає масив об'єктів, відповідних запущеним в системі процесам. Другий командлет ***Sort-Object*** впорядковує об'єкти, які проходять конвеєром, за значенням властивості ***WS*** (обсяг пам'яті, займаної процесом). Нарешті, третій командлет ***Select-Object*** вибирає з упорядкованого масиву об'єкта останні п'ять елементів.

Припустимо тепер, що нам потрібно отримати список запущених в системі процесів, у якому були б вказані тільки імена, і ідентифікатори. Якщо ви не пам'ятаєте назви потрібних властивостей, то можна за допомогою командлета ***Get-Member*** знову переглянути структуру об'єктів, що повертаються командою ***Get-Process***:

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

```
Name                   MemberType     Definition
----                   ----------     ----------
BasePriority           Property       System.Int32 BasePriority {get;}
Id                     Property       System.Int32 Id {get;}
ProcessName            Property       System.String ProcessName {get;}
```

Отже, в підсумкових об’єктах нам потрібно залишити тільки властивості ***ProcessName*** і ***Id***. Це можна зробити, вказавши імена потрібних властивостей у якості параметрів командлет ***Select-Object***:

```
PS C:\> Get-Process | Select-Object ProcessName, Id

ProcessName                                                    Id
-----------                                                    --
AcroRd32                                                     5808
AcroRd32                                                     5952
AGSService                                                   1364
-------------------------------------------------------------------
```

Використання змінних

У змінних зберігаються всі можливі значення, навіть якщо вони є об’єктами. Імена змінних в **PowerShell** завжди повинні починатися з символу «***$***». Можна зберегти список процесів у змінної, це дозволить у будь-який час отримувати доступ до списку процесів. Присвоїти значення змінної не важко:

```
$a = Get-Process | Sort-Object CPU
```

Вивести вміст змінної можна, надрукувавши в командному рядку ***$a***.

Виконання довільних дій над об’єктами у конвеєрі

Командлет ***ForEach-Object*** дозволяє виконати певний блок сценарію (код на мові **PowerShell**) для кожного об’єкта в конвеєрі. Іншими словами, за допомогою цього командлета можна виробляти довільні операції над елементами конвеєра. Для прикладу давайте підрахуємо загальний обсяг файлів, що зберігаються у певному каталозі диска. Для цього оголосимо змінну ***$TotalLength*** і присвоїмо її значення нулю:

```
PS D:\Gleb> $TotalLength=0
```

Тепер виконаємо команду ***dir*** і результат її роботи передамо конвеєром командлету ***ForEach-Object***:

```
PS D:\Gleb> dir | ForEach-Object {$TotalLength+=$_.Length}
```

У блоці сценарію командлет ***ForEach-Object*** до поточного значення змінної ***$TotalLength*** додає значення властивості ***Length***, що проходить через конвеєр об’єкта (розмір відповідного цьому об’єкту файлу). У результаті у змінній ***$TotalLength*** буде зберігатися загальний розмір файлів в байтах:

```
PS D:\Gleb>  $TotalLength
20336777
```

Командлет Group-Object – групування об’єктів

Об’єкти, що проходять конвеєром, можна згрупувати за значенням певних властивостей за допомогою командлета ***Group-Object***. В одну групу будуть потрапляти ті, що мають однакові значення зазначених властивостей (властивості можуть бути обчислювані).

Розглянемо приклад. Командлет ***Get-Process*** генерує об’єкти, що мають властивості ***Company*** (назва компанії-розробника певного модуля, запущеного в операційній системі в якості процесу). Виконаємо групування цих об’єктів за значенням властивості ***Company***:

```
PS C:\> Get-Process | Group-Object Company

Count Name                      Group
----- ----                      -----
   94                           {System.Diagnostics.Process (AGSService)
    1 Avast Software s.r.o.     {System.Diagnostics.Process (AvastUI)}
    6 Google Inc.               {System.Diagnostics.Process (chrome)…
------------------------------------------------------------------------
```

Завдання

Для виконання завдань можливо знадобиться дізнатися назви властивостей відповідних об’єктів, що повертаються тим чи іншим командлетом!

1. Вивести інформацію про 10 процесів, які споживають найбільший процесорний час.
2. Створити список усіх служб (командлет ***Get-Service***) та відсортувати їх за статусом.
3. Вивести всі працюючі на цей момент служби, назва яких починається на літеру «S».
4. Вивести список файлів поточного каталогу, залишивши тільки його назву та розмір.
5. Вивести інформацію про те, скільки файлів, що мають відповідне розширення, є у поточному каталозі, та відсортувати її.
6. Видалити у поточному каталозі всі файли, що мають розмір менше ніж 4Кб.

Контрольні питання

1. Яким чином використовуються конвеєри у PowerShell?
2. Для чого призначений командлет Where-Object?
3. Яким чином задається умова для перевірки у командлеті Where-Object?
4. Які оператори порівняння існують у PowerShell? Назвіть їх.

5. Які логічні оператори існують у PowerShell? Назвіть їх.
6. Який командлет призначений для сортування об’єктів? Як задати режим сортування об’єктів?

Лабораторна робота № 11. Форматування результату виведення команд у Windows PowerShell. Збереження даних у файл

Розглянемо, яким чином система формує рядки тексту, які виводяться на екран в результаті виконання тієї чи іншої команди (нагадаємо, що командлети **PowerShell** повертають .NET-об'єкти, які, як правило, не знають, яким чином відображати себе на екрані).

У **PowerShell** є база даних (набір XML-файлів), що містить модулі форматування за замовчуванням для різних типів .NET-об'єктів. Вони визначають, які властивості об'єкта відображаються під час виведення і в якому форматі: списку або таблиці. Коли об'єкт досягає кінця конвеєра, **PowerShell** визначає його тип і шукає його в списку об'єктів, для яких визначено правило форматування. Якщо такий тип в списку виявлений, то до об'єкта застосовується відповідний модуль форматування; якщо немає, то **PowerShell** лише відображає властивості цього .NET-об'єкта.

Також у **PowerShell** можна задавати правила форматування інформації, що виводиться командлетами, і подібно командному інтерпретатору **Cmd.exe** перенаправляти її в файл, на принтер або в порожній пристрій.

Форматування інформації, що виводиться

У традиційних оболонках команди і утиліти самі форматують виведену інформацію. Деякі команди (наприклад, ***dir*** в інтерпретаторі **Cmd.exe**) дозволяють налаштовувати формат виведення за допомогою спеціальних параметрів.

В оболонці **PowerShell** висновок форматують тільки чотири спеціальні командлети ***Format*** (табл. 14). Це спрощує вивчення, оскільки не потрібно запам'ятовувати засоби і параметри форматування для інших команд (інші командлети виведення не форматують).

Таблиця 14

Командлети PowerShell для форматування виведення

Командлет	Опис
Format-Table	Форматує виведення команди у вигляді таблиці, стовпці якої містять властивості об'єкта (також можуть бути додані обчислювані стовпці). Підтримується можливість групування інформації, що виводиться

Закінчення таблиці

Format-List	Виведення форматується як список властивостей, у якому кожна властивість відображається на новому рядку. Підтримується можливість групування інформації, що виводиться
Format-Custom	Для форматування виведення використовується представлення (view)
Format-Wide	Форматує об'єкти у вигляді широкої таблиці, у якій відображається тільки одна властивість кожного об'єкта

Як уже зазначалося, якщо жоден з командлетів ***Format*** чітко не вказаний, то використовується модуль форматування за замовчуванням, який визначається за типом інформації, що відображається. Наприклад, під час виконання командлету ***Get-Service*** інформація за замовчуванням виводиться як таблиця з трьома стовпцями (***Status***, ***Name*** і ***DisplayName***):

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_                  .NET Runtime Optimization
optimizatio                   Service v...
Stopped    COMSysApp          Системное приложение COM+
Running    CryptSvc           Службы криптографии
Running    DcomLaunch         Запуск серверных процессов
                              DCOM
Running    Dhcp               DHCP-клиент
...
```

Для зміни формату виведеної інформації потрібно направити її конвеєром відповідному командлету ***Format***. Наприклад, наступна команда виведе список служб за допомогою командлета ***Format-List***:

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
```

Під час використання формату списку виводиться більше відомостей про кожну службу, ніж в форматі таблиці (замість трьох стовпців відомостей про кожну службу в форматі списку виводяться дев'ять рядків). Однак це зовсім не означає, що командлет ***Format-List*** витягує додаткові відомості про служби. Вони містяться в об'єктах, які повернуться командлетом ***Get-Service***, проте командлет ***Format-Table***, який буде використовуватися за замовчуванням, відкидає їх, тому що не може вивести на екран більше трьох стовпців.

У процесі форматування виведення за допомогою командлетів ***Format-List*** і ***Format-Table*** можна вказувати імена властивостей об'єкта, які повинні бути відображені (нагадаємо, що переглянути список властивостей, що має об'єкт, дозволяє розглянутий раніше командлет ***Get-Member***). Наприклад:

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
. . .

Вивести всі наявні в об'єктів властивості можна за допомогою параметра *******, наприклад:

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

Перенаправлення виведеної інформації

В оболонці **PowerShell** є кілька командлетів, за допомогою яких можна керувати виведенням даних. Ці командлети починаються зі слова ***Out***, їх список можна отримати за допомогою командлета:

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

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

За замовчуванням виводиться інформація, передається командлету ***Out-Default***, який в свою чергу, делегує всю роботу з виведення рядків на екран командлету ***Out-Host***. Для розуміння цього механізму потрібно враховувати, що архітектура **PowerShell** має на увазі відмінність між, власне, ядром оболонки (інтерпретатором команд) і головним додатком (**host**), який його використовує. У якості головного може виступати будь-який додаток, у якому реалізована низка спеціальних інтерфейсів, що дозволяють коректно інтерпретувати отримувану від **PowerShell** інформацію. У нашому випадку головним додатком є консольне вікно, у якому ми працюємо з оболонкою, і командлет **Out-Host**, що передає інформацію, яка виводиться в консольне вікно.

Параметр **Paging** командлета ***Out-Host***, подібно команді ***more*** інтерпретатора **Cmd.exe**, дозволяє організувати посторінкове виведення інформації, наприклад:

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

Збереження інформації у файлі

Командлет ***Out-File*** дозволяє направити виведену інформацію замість вікна консолі у текстовий файл. Аналогічне завдання вирішує оператор перенаправлення (>), проте командлет ***Out-File*** має кілька додаткових параметрів, за допомогою яких можна більш гнучко керувати виведенням: задавати тип кодування файлу (параметр ***-Encoding***), задавати довжину виведених рядків у знаках (параметр ***-Width***), вибирати режим додавання до файлу (параметр ***-Append***). Додати інформації у кінець файлу можливо і традиційним способом з використанням оператора перенаправлення (>>).

Наприклад, наступна команда направить інформацію про процеси, що виконуються на комп'ютері в файл ***C:\Process.txt***, причому зазначений файл буде записаний в форматі ASCII:

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

Придушення виведення

Командлет ***Out-Null*** служить для поглинання будь-яких своїх вхідних даних. Це може стати в нагоді для придушення виведення на екран непотрібних відомостей, отриманих у якості побічного ефекту виконання будь-якої команди. Наприклад, у процесі створення каталогу командою ***mkdir*** на екран виводиться його вміст:

```
PS C:\> mkdir spo
Каталог: Microsoft.PowerShell.Core\FileSystem::C:\
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          03.01.2009 1:01                spo
```

Якщо ці відомості не потрібні, то результат виконання команди ***mkdir*** необхідно передати конвеєром командлету ***Out-Null***:

mkdir spo | Out-Null

Завдання

1. Вивести інформацію щодо процесів, що мають Id, який більше за 7000, у табличному вигляді, залишивши лише тільки Id процесу, його ім'я та кількість процесорного часу, що зайняв процес. Результат вивести у відсортованому вигляді за спаданням.
2. Вивести інформацію про файли, що знаходяться у кореневому каталозі Вашого мережевого диску у вигляді списку, залишивши тільки наступні властивості: ім'я, розмір, дата останньої зміни. Результат вивести у відсортованому вигляді за розширенням файлу.

3. Вивести інформацію про підкаталоги, що знаходяться у кореневому каталозі вашого мережевого диску у вигляді списку, залишивши наступні властивості: ім'я, дата створення, дата та час останнього звернення до каталогу. Результат вивести у відсортованому вигляді за датою створення.
4. Змінити завдання 3 наступним чином: залишити тільки підкаталог, який був створений раніше за всі інші, вивести інформацію про нього у найбільш детальному вигляді (за допомогою командлету ***Format-Custom***). Що було отримано у результаті? Чи є зручним таке виведення?
5. Змінити завдання 4 так, щоб інформація, що виводиться командлетом Format-Custom виводилася з можливістю її посторінкового перегляду.
6. Змінити завдання 4 так, щоб виводилася детальна інформація тільки на першому рівні вкладення об'єктів (за допомогою параметра ***-Depth*** зі значенням ***1***).
7. Вивести інформацію про зупинені системні служби у файли ***stopped1.txt*** та ***stopped2.txt*** двома різними способами.
8. Додати до існуючих файлів ***stopped1.txt*** та ***stopped2.txt*** двома різними способами інформацію про працюючі служби у системі.
9. Створити у кореневому каталозі вашого мережевого диску новий файл file1.txt, не показуючи результат.
10. Видалити файл file1.txt, не показуючи результат.

Контрольні питання

1. Які командлети призначені для форматування виведеної інформації? У чому полягає їх різниця один від одного?
2. Які командлети призначені для перенаправлення інформації? У які пристрої можна її перенапрявляти? Наведіть приклади.
3. Куди перенапрявляє інформацію командлет ***Out-Host***?
4. Який командлет дозволяє записати виведену інформацію у файл?
5. Для чого призначений командлет ***Out-Null***?

Лабораторна робота № 12. Використання регулярних виразів у Windows PowerShell

Регулярні вирази – призначення та використання

Регулярні вирази (або скорочено «регекспи» (regexp, regular expressions)) володіють більшою силою, і здатні значно спростити життя системного адміністратора або програміста. У **PowerShell** регулярні вирази легкодоступні, зручні у використанні і максимально функціональні. **PowerShell** використовує реалізацію регулярних виразів .NET.

Регулярні вирази – це спеціальна міні-мова, службовець для розбору (parsing) текстової інформації. З її допомогою можна розділяти рядки на компоненти, вибирати потрібні частини рядків для подальшої обробки, робити заміни тощо.

Знайомство з регулярними виразами почнемо з більш простої технології, що служить подібним цілям – з групових символів. Напевно ви не раз виконували команду ***dir***, вказуючи їй як аргумент маску файлу, наприклад ****.exe***. У цьому випадку зірочка означає «будь-яку кількість будь-яких символів». Аналогічно можна використо-вувати і знак питання, він буде означати «один будь-який символ», тобто ***dir ??.exe*** виведе всі файли з розширенням ***.exe*** і ім'ям з двох символів. У **PowerShell** можна застосовувати і ще одну конструкцію – групи символів. Так наприклад ***[a-f]*** означатиме «один будь-який символ від ***a*** до ***f***, тобто ***(a, b, c, d, e, f)***», а ***[smw]*** будь-яку з трьох букв (***s***, ***m*** або ***w***). Таким чином команда ***get-childitem [smw]??.exe*** виведе файли з розширенням ***.exe***, у яких ім'я складається з трьох букв, перша буква або ***s***, або ***m***, або ***w***.

Оператор PowerShell -match

На початку вивчення ми будемо використовувати оператор **PowerShell *-match***, який дозволяє порівнювати текст ліворуч від нього, з регулярним виразом праворуч. У разі якщо текст підпадає під регулярний вираз, оператор видає ***True***, інакше - ***False***.

PS C:\> «PowerShell» -match «Power»
True

Під час порівняння з регулярним виразом шукаємо лише входження рядка, повний збіг тексту необов'язково (зрозуміло, це можна змінити). Тобто достатньо, щоб регулярний вираз зустрічався в тексті.

PS C:\> «Shell» -match «Power»
False

PS C:\> «PowerShell» -match «rsh»
True

Ще одна особливість: оператор ***-match*** за замовчуванням не чутливий до регістру символів (як і інші текстові оператори в **PowerShell**), якщо ж потрібна чутливість до регістру, використовується ***-cmatch***:

PS C:\> «PowerShell» -cmatch «rsh»
False

Використання груп символів

У регулярних виразах можна використовувати і групи символів:

PS C:\> Get-Process | Where-Object {$_.Name -match «sy[ns]»}

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

Діапазони в цих групах:

PS C:\> «яблуко», «апельсин», «груша», «абрикос» -match «а[а-п]»
апельсин
абрикос

У лівій частині оператора ***-match*** знаходиться масив рядків, і оператор відповідно вивів лише ті рядки, які підійшли під регулярний вираз.

Набагато цікавіше використовувати діапазони для визначення цілих класів символів. Наприклад ***[а-я]*** означатиме будь-яку букву російського алфавіту, а ***[a-z]*** англійського. Аналогічно можна чинити з цифрами – наступна команда виведе всі процеси, в іменах яких зустрічаються цифри:

PS C:\> Get-Process | Where-Object {$_.Name -match «[0-9]»}

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

Оскільки ця група використовується достатньо часто, для неї була виділена спеціальна послідовність – ***\d*** (від слова **digit**). За змістом вона повністю ідентична ***[0-9]***, але коротша.

PS C:\> Get-Process | Where-Object {$_.Name -match «\d»}

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

Так само послідовність була виділена для групи «будь-які літери будь-якого алфавіту, будь-які цифри, або символ підкреслення» ця група позначається як ***\w*** (від **word**) вона приблизно еквівалентна конструкції ***[a-za-я_0-9]*** (в ***\w*** ще входять символи інших алфавітів які використовуються для написання слів).

Інша популярна група: ***\s*** – «пробіл, або інший символ пробілу» (наприклад символ табуляції). Скорочення від слова **space**. У більшості випадків ви можете позначати пробіл просто як пробіл, але ця конструкція додає читабельності регулярному виразу.

Не менш популярною групою можна назвати символ **.** (**точка**). Точка в регулярних виразах є аналогічною за змістом знаку питання в підстановлювальних символах, тобто позначає один будь-який символ.

Усі перераховані вище конструкції можна використовувати як окремо, так і в складі груп, наприклад ***[\s \d]*** буде відповідати будь-якій цифрі або пробілу. Якщо ви хочете вказати всередині групи символ **-** (тире/мінус) то треба або екранувати його символом \ (зворотний слеш), або поставити його на початку групи, щоб він не був випадково витлумачений як діапазон:

PS C:\> «?????», «Word», «123», «-» -match «[-\d]»
123
-

Негативні групи і якоря

Розглянемо деякі більш «продвинуті» конструкції регулярних виразів.

Передбачається, що ви вже знаєте, як вказати регулярному виразу, які символи і/або їх послідовності повинні бути в рядку для збігу. А що якщо потрібно вказати не ті символи, які мають бути присутніми, а ті яких не повинно бути? Тобто якщо потрібно вивести лише приголосні букви, ви можете їх перерахувати, а можете використовувати і негативну групу з голосними, наприклад:

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

«Кришка» у якості першого символу групи символів означає саме **заперечення**. Тобто на місці групи може бути присутнім будь-який символ окрім перерахованих. Для того щоб включити заперечення в символьних групах (***\d***, ***\w***, ***\s***), не обов'язково укладати їх у квадратні дужки, досить перевести їх у верхній регістр. Наприклад ***\D*** буде означати «що завгодно, крім цифр», а ***\S*** «все крім пробілів»:

PS C:\> «a»,»b»,»1»,»c»,»45» -match «\D»
a
b
c

PS C:\> «a»,»-»,»*»,»c»,»&» -match «\W»
-
****
&

Символьні групи дозволяють вказати лише вміст однієї позиції, один символ, що знаходиться у невизначеному місці рядка. А якщо треба наприклад вибрати всі слова, які починаються з літери ***w***? Якщо просто помістити цю букву в регулярний вираз, то воно співпаде для всіх рядків, де ***w*** взагалі зустрічається, і не важливо – на початку, всередині або вкінці рядка. У таких випадках на допомогу приходять «якоря». Вони дозволяють проводити порівняння, починаючи з певної позиції в рядку.

^ (Кришка) є якорем початку рядка, а ***$*** (знак долара) – позначає кінець рядка.

Не заплутайтеся – ^ як символ заперечення використовується лише на початку групи символів, а за межами групи – цей символ є вже якорем. Авторам регулярних виразів явно не вистачало спеціальних символів, і вони за можливості використовували їх більш ніж в одному місці.

Приклад. Виведення списку процесів, імена яких починаються з літери ***w***:

PS C:\> Get-Process | Where-Object {$_.Name -match «^w»}

```
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
```

Ця команда вивела процеси, у яких відразу після початку імені (^) слідує символ ***w***. Інакше кажучи, ім'я починається на ***w***. Для ускладнення прикладу, і для спрощення розуміння, додамо сюди «кришку» в значенні негативної групи:

```
PS C:\> Get-Process | Where-Object {$_.Name -match «^w[^l-z]»}

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
```

Тепер команда вивела процеси, у яких ім'я починається з символу ***w***, а наступний символ є чим завгодно, тільки не символом з діапазону ***l-z***.

Для закріплення випробуємо другий якір – кінець рядка:

```
PS C:\> «Яблука», «Груші», «Диня», «Єнот», «Апельсини», «Персик» -match «[аui] $»
Яблука
Груші
Апельсини
```

Цей вислів вивів нам всі слова у яких остання буква ***а***, ***и*** або ***і***.

Якщо ви можете точно описати вміст всього рядка, то ви можете використовувати і обидва якоря одночасно:

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

Це регулярний вираз виводить всі рядки, які починаються з літери ***а***, за якою слідує один будь-який символ (точка), потім символ ***с*** або ***b*** і потім кінець рядка.

Позначення деяких класів символів (метасимволів) наведені в таблиці 15.

Таблиця 15

Метасимволи, використовувані в регулярних виразах

Метасимвол	Опис метасимвола
.(точка)	Припускає, що в кінцевому виразі на її місці буде стояти будь-який символ. Продемонструємо це на прикладі набору англійських слів: *Вихідний набір рядків:* ***wake*** ***make*** ***machine*** ***cake*** ***maze*** *Регулярний вираз:* ***ma.e*** *Результат:* ***make*** ***maze***
\w	Заміщує будь-які символи, які відносяться до букв, цифр і знаку підкреслення. Приклад: *Вихідний набір рядків:* ***abc*** ***a$c*** ***a1c*** ***ac*** *Регулярний вираз:* ***a\wc*** *Результат:* ***abc*** ***a1c***
\W	Заміщує всі символи, крім букв, цифр і знака підкреслення (тобто є зворотним метасимвол ***\w***). Приклад: *Вихідний набір рядків:* ***abc*** ***a$c*** ***a1c*** ***a c*** *Регулярний вираз:* ***a\Wc*** *Результат:* ***a$c a c***
\d	Заміщує всі цифри. Продемонструємо його дію на тому ж прикладі: *Вихідний набір рядків:* ***abc*** ***a$c*** ***a1c***

	a c *Регулярний вираз:* ***a\dc*** *Результат:* ***a1c***
\D	Заміщує всі символи, крім цифр, наприклад: *Вихідний набір рядків:* ***abc*** ***a$c*** ***a1c*** ***a c*** *Регулярний вираз:* ***a\Dc*** *Результат:* ***abc*** ***a$c*** ***a c***

Завдання

1. Вивести список запущених служб, назва яких починається зі слів «***MS***» (формат виведення обрати на власний розсуд).
2. Знайти у каталозі ***C:\Windows\Microsoft.NET*** та всіх його підкаталогах файли динамічних бібліотек (***.dll***), назви яких починаються на «***aspnet***».
3. Знайти у кореневому каталозі вашого мережевого диску та його підкаталогах файли зображень, що мають розширення ***.jpg*** та містять у своїй назві тільки літери українського алфавіту.
4. Знайти у кореневому каталозі вашого мережевого диску та його підкаталогах файли, в іменах яких у якості другого символу використовується літера «***a***»
5. Знайти у каталозі ***C:\Windows*** конфігураційні файли, в іменах яких присутні числа.
6. Вивести список команд, які тим чи іншим чином стосуються роботи з процесами (їх імена закінчуються на слово «***Process***»).

Контрольні питання

1. Для чого призначені регулярні вирази?
2. Які дії виконує оператор ***–match*** у **PowerShell**? У чому різниця операторів ***–match*** та ***–cmatch***?
3. Яким чином задаються групи символів у регулярних виразах?
4. Які символи у регулярних виразах позначають відповідно початок та кінець рядка?
5. Які метасимволи використовуються у регулярних виразах PowerShell? Наведіть приклади.

Лабораторна робота № 13. Програмування сценаріїв Windows PowerShell. Керуючі конструкції мови PowerShell: оператори перевірки умов, цикли. Використання масивів

Змінні PowerShell

Як ми вже знаємо, імена змінних **PowerShell** завжди починаються зі знака долара (***$***). Змінні **PowerShell** не потрібно попередньо оголошувати або описувати, вони створюються у процесі першого присвоєння значення змінній. Якщо спробувати звернутися до неіснуючої змінної, то система поверне значення ***$Null***.

$Null, ***$True*** та ***$False***, є спеціальними змінними, визначеними в системі. Змінити значення цих змінних не можна.

Призначена для користувача змінна створюється після першого присвоювання їй значення. Наприклад, створимо цілочисельну змінну ***$a***:

PS C:\> $a=1
PS C:\> $a
1

Дати користувачу можливість ввести інформацію з екрана та присвоїти результат введення у певну змінну дозволяє командлет ***Read-Host***.

PS C:\> $name=Read-Host «Input your name»
Input your name: Gleb
PS C:\> $name
Gleb

Керуючі інструкції

Інструкція If …ElseIf … Else

У загальному випадку синтаксис інструкції ***If*** має вигляд:

```
If (умова1)
    {Блок_коду1}
[ElseIf (умова2)]
    {Блок_коду2}]
[Else
    {Блок_коду3}]
```

Під час виконання інструкції ***If*** перевіряється істинність умовного виразу ***умова1***.

Якщо ***умова1*** має значення ***$True***, то виконується ***блок_коду1***, після чого виконання інструкції ***if*** завершується. Якщо ***умова1*** має значення ***$False***, перевіряється істинність умовного виразу ***умова2***. Якщо ***умова2*** має значення ***$True***, то виконується ***блок_коду2*** і виконання інструкції ***if*** завершується. Якщо і ***умова1***, і ***умова2*** мають значення ***$False***, то виконується ***блок_коду3*** і виконання інструкції ***if*** завершується.

Приклад використання інструкції ***if*** в інтерактивному режимі роботи. Спочатку змінній ***$a*** дамо значення 10:

PS C:\> $a = 10

Потім порівняємо значення змінної з числом 15:

PS C: \> If ($a -eq 15) {
>> 'Значення $ a дорівнює 15'
>>}
>> Else { 'Значення $ a не дорівнює 15'}
>>
Значення $ a не дорівнює 15

Цикли While и Do … While

Найпростіший з циклів PS – цикл ***While***, у якому команди виконуються допоки умова, що перевіряється, має значення ***$True***. Інструкція ***While*** має наступний синтаксис:

While (умова) {блок_команд}

Цикл ***Do ... While*** схожий на цикл ***While***, однак умова в ньому перевіряється не до блоку команд, а після: ***Do {блок_команд} While (умова)***. Наприклад:

PS C:\> $val=0
PS C:\>Do {$val++; $val} While ($val –ne 3)
1
2
3

Цикл For

Зазвичай цикл ***For*** застосовується для проходження масивом і виконанням певних дій з кожним із його елементів. Синтаксис інструкції ***For***:

For (ініціалізація; умова; повторення) {блок_команд}

Приклад:

PS C:\> For ($i=0; $i –lt 3; $i++) {$i }

0

1

2

Цикл ForEach

Інструкція ***ForEach*** дозволяє послідовно перебирати елементи колекцій. Найпростіший тип колекції – масив. Особливість циклу ***ForEach*** полягає у тому, що його синтаксис і виконання залежать від того, де розташована інструкція ***ForEach***: поза конвеєром команд або всередині.

Інструкція ForEach поза конвеєром команд

У цьому випадку синтаксис циклу ***ForEach*** має вигляд:

ForEach ($елемент in $колекція) {блок_команд}

Під час виконання циклу ***ForEach*** автоматично створюється змінна ***$елемент***. Перед кожною ітерацією в циклі цієї змінної присвоюється значення чергового елемента в колекції. У розділі ***блок_команд*** містяться команди, які виконуються на кожному елементі колекції. Наведений нижче цикл ***ForEach*** відображає значення елементів масиву ***$lettArr***:

PS C:\> $lettArr = “a”, “b”,”c”

PS C:\> ForEach ($lett in $lettArr) {Write-Host $lett}

a

b

c

Інструкція ***ForEach*** може також використовуватися спільно з командлетами, які повертають колекції елементів, наприклад:

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

У прикладі створюється і обнуляється змінна ***$ln***, потім у циклі ***ForEach*** за допомогою командлета ***dir*** формується колекція файлів з розширенням ***txt***, що знаходяться у поточному каталозі. Інструкція ***ForEach*** перебирає всі елементи цієї колекції, на кожному кроці до поточного файлу виконується звернення за допомогою змінної ***$f***. У блоці команд циклу ***ForEach*** до поточного значення змінної ***$ln*** додається значення властивості ***length*** (розмір файлу) змінної ***$f***. У результаті виконання циклу в змінній ***$ln*** буде отримано сумарний розмір файлів у поточному каталозі, які мають розширення ***txt.***

Інструкція ForEach всередині конвеєра команд

Якщо інструкція ***ForEach*** з'являється всередині конвеєра команд, то PS використовує псевдонім ***ForEach***, відповідний командлету ***ForEach-Object***. У цьому випадку фактично виконується командлет ***ForEach-Object*** і не потрібна частина інструкції ***($елемент in $колекція)***, оскільки елементи колекції блоку команд надає попередня команда конвеєра.

Синтаксис інструкції ***ForEach*** всередині конвеєра команд має вигляд:

команда | ForEach {блок_команд}

Розглянутий вище приклад підрахунку сумарного розміру файлів з поточного каталогу для цього варіанта інструкції ***ForEach*** прийме наступний вигляд:

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

У наведеному прикладі спеціальна змінна ***$_*** використовується для звернення до поточного об'єкту конвеєра і вилучення його властивостей.

Створення і використання масивів

Для створення та ініціалізації масиву достатньо привласнити значення його елементів. Значення, що додаються в масив, розділяються комами і відокремлюються від імені масиву символом присвоювання. Наприклад, наступна команда створить масив ***$a*** з трьох елементів:

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

Можна створити та ініціалізувати масив, використовуючи оператор діапазону (..). Наприклад, команда:

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

створює та ініціалізує масив ***$b***, що містить ***6*** значень ***10***, ***11***, ***12***, ***13***, ***14*** і ***15***.

Для створення масиву може використовуватися операція введення значень його елементів з текстового файлу:

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

У наведеному прикладі результат виконання командлета ***Get-Content*** присвоюється масиву ***$f***. Необов'язковий параметр ***-TotalCount*** обмежує кількість прочитаних елементів величиною ***25***. Властивість об'єкта масив ***length*** – має значення, що дорівнює кількості елементів масиву, в прикладі воно дорівнює ***25*** (передбачається, що в текстовому файлі ***munb.txt*** принаймні ***25*** рядків).

Звернення до елементів масиву

Довжина масиву (кількість елементів) зберігається у властивості ***Length***. Для звернення до певного елементу масиву потрібно вказати його індекс в квадратних дужках після імені змінної. Нумерація елементів масиву завжди починається з нуля. У якості індексу можна вказувати і негативні значення, відлік буде вестися з кінця масиву – індекс ***-1*** відповідає останньому елементу масиву.

Операції з масивами

За замовчуванням масиви **PowerShell** можуть містити елементи різних типів (цілі 32-х розрядні числа, рядки, дійсні та інші), тобто є поліморфними. Можна створити масив з жорстко заданим типом, що містить елементи тільки одного типу, вказавши потрібний тип в квадратних дужках перед ім'ям змінної. Наприклад, наступна команда створить масив 32-х розрядних цілих чисел:

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

Масиви **PowerShell** базуються на .NET-масивах, що мають фіксовану довжину, тому звернення до інших елементів масиву фіксується як помилка. Є спосіб збільшення первісно визначеної довжини масиву. Для цього можна скористатися оператором конкатенації + або +=. Наприклад, наступна команда додасть до масиву ***$a*** два нові елементи зі значеннями ***5*** і ***6***:

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

Під час виконання оператора += відбувається наступне:

– створюється новий масив, розмір якого достатній для переміщення в нього всіх елементів;

– початковий вміст масиву копіюється в новий масив;

– нові елементи копіюються в кінець нового масиву.

Таким чином, насправді створюється новий масив більшого розміру.

Можна об'єднати два масиви, наприклад ***$b*** і ***$c*** в один за допомогою операції конкатенації +. Наприклад:

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

Функції у PowerShell

Функція в **PowerShell** – це блок коду, який має назву і знаходиться в пам'яті до завершення поточного сеансу командної оболонки. Якщо функція визначається без формальних параметрів, то для її завдання досить вказати ключове слово **Function**, потім ім'я функції і список виразів, що складають тіло функції (такий список повинен бути заключений у фігурні дужки). Наприклад, створимо функцію ***MyFunc***:

PS C:\> Function MyFunc {«Всім привіт!»}

Для виклику цієї функції достатньо лише ввести її ім'я:

PS C:\> MyFunc

Всім привіт!

Функція в **PowerShell** має доступ до аргументів, з якими вона була запущена, навіть якщо під час визначення цієї функції не були задані формальні параметри. Усі аргументи, з якими була запущена функція, автоматично зберігаються в змінній ***$Args***. Іншими словами, в змінній ***$Args*** міститься масив, елементами якого є параметри функції, зазначені в ході її запуску. Для прикладу додамо змінну ***$Args*** в нашу функцію ***MyFunc***:

PS C:\> Function MyFunc {«Привіт, $Args!»}

Оскільки змінна ***$Args*** поміщена в рядок у подвійних лапках, то під час запуску функції значення цієї змінної буде обчислене (розширене), і результат буде вставлений в рядок. Викличемо функцію ***MyFunc*** з трьома параметрами:

PS C:\> MyFunc Андрій Сергій Іван

Привіт, Андрій Сергій Іван!

У **PowerShell**, як і в більшості інших мов програмування, у процесі опису функції можна задати список формальних параметрів, значення яких під час виконання функції будуть замінені значеннями фактично переданих аргументів.

Список формальних параметрів вказується в круглих дужках після імені функції. Визначимо, наприклад, функцію ***Subtract*** для знаходження різниці двох своїх аргументів (зменшуваному відповідає параметр ***$From***, від'ємнику – параметр ***$Count***):

PS C:\> Function Substract($From, $Count) {$From-$Count}

У процесі виконання функції ***Subtract*** її формальні параметри будуть замінені фактичними аргументами, які визначаються за позицією в командному рядку, або за іменем.

Наприклад:

PS C:\> Substract 10 2

8

У цьому випадку відповідність формальних параметрів фактично переданих аргументів визначається за позицією: замість першого параметра ***$From*** підставляється число ***10***, замість другого ***$Count*** число ***2***.

У ході зазначення аргументів можна використовувати імена формальних параметрів (порядок зазначення аргументів у цьому разі стає несуттєвим), наприклад:

PS C:\> Substract -From 10 -Count 2

8

PS C:\> Substract -Count 3 -From 5

2

Сценарії PowerShell

Сценарії **PowerShell** є текстовими файлами з розширенням ***ps1***, у яких записаний код (команди, оператори та інші конструкції) на мові **PowerShell**. На відміну від сценаріїв **WSH** і командних файлів інтерпретатора **cmd.exe**, сценарії **PowerShell** можна писати поетапно, безпосередньо в самій оболонці, переміщаючи потім готовий код у зовнішній текстовий файл. Такий підхід значно спрощує вивчення мови і налагодження сценаріїв, дозволяючи відразу бачити результат виконання його окремих частин.

Сценарії виконуються системою тільки у тому випадку, коли це дозволено поточною політикою виконання. За замовчуванням діє політика **Restricted**, яка повністю забороняє виконання сценаріїв **PowerShell**. Це зроблено з міркувань безпеки, тому що в сценаріях може міститися шкідливий код, який може пошкодити систему або несанкціоновано скористатися одними даними.

Перевірити активну політику виконання за допомогою командлету ***Get-ExecutionPolicy***.

PS C:\Script> Get-ExecutionPolicy
Restricted

Якщо в системі діє більш сувора політика безпеки (***Restricted*** або ***AllSigned***), потрібно встановити політику ***RemoteSigned***, яка дозволяє виконувати непідписані локальні сценарії:

PS C:\Script> Set-ExecutionPolicy RemoteSigned

Цей командлет може не спрацювати, якщо запускати його не від імені адміністратора. Для того, щоб дозволити виконання сценаріїв для поточного користувача, необхідно додати параметр ***–Scope*** зі значенням ***CurrentUser***:

PS C:\Script> Set-ExecutionPolicy -Scope CurrentUser RemoteSigned

Під час запуску сценаріїв **PowerShell** шлях до файлу з кодом потрібно завжди вказувати чітко, навіть якщо сценарій знаходиться у поточному каталозі, оскільки це запобігає можливому несанкціонованому запуску іншої виконуваної програми з аналогічним ім'ям, що знаходиться, наприклад, в системному каталозі. У цьому разі можна навіть не вказувати розширення ***.ps1***. А можна вказати і повний шлях до файлу.

Розбір і обробка аргументів, переданих в сценарії, проводиться практично так само, як і у функціях (взагалі, сценарій – це фактично функція, яка знаходиться не в оперативній пам'яті, а на диску).

Аргументи вказуються після імені сценарію і розділяються між собою пробілами. Змінна ***$Args*** всередині сценарію містить масив, елементами якого є аргументи функції, зазначені в ході її запуску. Для прикладу напишемо сценарій ***SumArgs.ps1***. який буде повідомляти кількість параметрів, з якими він запущений, і їх суму.

Файл **SumArgs.ps1**:

```
«Кількість аргументів: $($Args.count)»
$n=0
for($i=0; $i -lt $Args.Count; $i++) { $n+=$Args[$i] }
«Сума аргументів: $n»
```

```
PS C:\Script> .\SumArgs.ps1 4 7 8
Кількість аргументів: 3
Сума аргументів: 19
```

Як бачите, масив ***$Args*** в сценаріях має таке саме значення і обробляється так само, як у функціях.

У сценаріях можна визначати формальні параметри, замість яких під час виконання будуть підставлятися фактичні аргументи, передані в сценарій. У функціях формальні параметри перераховувалися в круглих

дужках після імені, тобто поза тілом функції. У сценаріях так вчинити не можна, оскільки тут весь вміст файлу є тілом сценарію, тому формальні параметри задаються за допомогою спеціальної інструкції ***Param***. Вона має бути найпершою командою у файлі, передувати їй можуть тільки порожні рядки і коментарі. Для прикладу напишемо сценарій ***Add.ps1*** з двома формальними параметрами, який буде виводити суму своїх аргументів.

Файл **Add.ps1**:

```
Param($x=2,$y=3)
$x+$y
```

Запустимо отриманий сценарій з аргументами і без них:

```
PS C:\Script> .\Add 10 20
30
PS C:\Script> .\Add
5
```

Усе працює, як і очікувалося: якщо не вказано жодних аргументів, то всередині сценарію використовуються значення за замовчуванням.

Завдання

1. Написати сценарій **PowerShell**, який знаходить у заданому користувачем каталозі один файл найбільшого розміру та три файли найменшого.
2. Написати сценарій **PowerShell** для перевірки наявності файлів документів (****.doc***, ****.txt***) у заданому користувачем каталозі. У разі позитивної відповіді – вивести їх список.
3. Написати сценарій **PowerShell**, що знаходить копії файлів у двох заданих користувачем каталогах. Для кожної такої пари вивести на екран запит про те, яку копію видалити.

Контрольні питання

1. Який синтаксис має оператор ***if*** у мові **PowerShell**? Наведіть приклади.
2. Як описати змінну в **PowerShell** та вивести її значення?
3. Які оператори циклів існують у **PowerShell**? Наведіть приклади.
4. Як оголосити масив у **PowerShell**? Які способи проініціалізувати масив існують? Наведіть приклади.
5. У чому полягає особливість функцій у мові **PowerShell**? Яким чином оголосити функції?
6. Які існують способи передачі у функцію у мові **PowerShell**?
7. Що представляє собою сценарії **PowerShell**? Які проблеми можуть виникнути під час його запуску?

Лабораторна робота № 14. Робота з COM- та .NET-об'єктами у сценаріях PowerShell

Робота з COM-об'єктами

Працюючи в **PowerShell**, можна ідентифікувати COM-об'єкти за їх програмними ідентифікаторами (**ProgID**) – символьними псевдонімами, що призначаються в ході реєстрації об'єктів в системі. Відповідно до загальноприйнятої угоди ідентифікатори **ProgID** мають такий вигляд: ***Бібліотека_типів.Клас.Версія*** або просто ***Бібліотека_типів.Клас***.

Достатньо часто ім'я бібліотеки типів збігається з ім'ям програми, що є сервером COM-об'єктів. Після точки в ProgID вказується ім'я класу, що містить властивості і методи COM-об'єкта, доступні для використання іншими додатками. Номер версії під час створення екземплярів COM-об'єктів, як правило, не використовується. Ось кілька прикладів **ProgID**: ***IntenetExplorer.Application*** (додаток **Internet Explorer**). ***Word.Application*** (додаток **Microsoft Word**), ***WScript.Shell*** (клас **Shell** з об'єктної моделі сервера сценаріїв **Windows Script Host**).

У **PowerShell** є командлет ***New-Object***, що дозволяє, зокрема, створювати екземпляри зовнішніх COM-об'єктів, вказуючи відповідний **ProgID** як значення параметра ***-ComObject***. Наприклад, екземпляр COM-об'єкту з програмним ідентифікатором ***WScript.Shell*** створюється наступним чином:

PS C:\> $Shell = New-Object -ComObject WScript.Shell

Подивимося, які властивості і методи є у COM-об'єкту ***WScript.Shell***. Для цього скористаємося, як зазвичай, командлетом ***Get-Member***, передавши йому конвеєром змінну ***$Shell***, у якій збережене посилання на зазначений COM-об'єкт:

PS C:\Users\Глеб> $Shell | Get-Member

```
TypeName: System._ComObject#{41904400-be18-11d3-a28b-00104bd35090}

Name                 MemberType Definition
----                 ---------- ----------
AppActivate          Method     bool AppActivate (Variant, Variant)
CreateShortcut       Method     IDispatch CreateShortcut (string)
Exec                 Method     IWshExec Exec (string)
---------------------------------------------------------------------------
```

Об'єктні моделі Microsoft Word і Excel

Одними з найпоширеніших і часто використовуваних серверів автоматизації у Windows є застосунки пакета Microsoft Office. Ми розглянемо на прикладах, яким чином можна виводити з **PowerShell** інформацію у дві найбільш поширені програми цього пакета: **Microsoft Word** і **Microsoft Excel**.

Хоча об'єктні моделі застосунків Microsoft Office досить складні, вони схожі одна на одну, причому для практичних цілей достатньо зрозуміти принцип роботи з декількома ключовими об'єктами.

На самому верхньому рівні об'єктної моделі **Word** знаходиться об'єкт ***Application***, який представляє безпосередньо сам додаток **Word** і містить (як властивості) всі інші об'єкти. Таким чином, об'єкт ***Application*** використовується для отримання доступу до будь-якого іншого об'єкту **Word**.

Сімейство ***Documents*** є властивістю об'єкта ***Application*** і містить набір об'єктів ***Document***, кожен з яких відповідає відкритому у **Word** документу. Клас об'єкт ***Document*** містить у якості своїх властивостей сімейства різних об'єктів документа: символів (***Characters***), слів (***Words***), речень (***Sentences***), параграфів (***Paragraphs***), закладок (***Bookmarks***) і т. ін.

Об'єктна модель **Excel** побудована за тим самим принципом, що і об'єктна модель **Word**. Основним об'єктом, що містить всі інші, є ***Application***. Окремі файли в **Excel** називають робочими книгами. Сімейство ***Workbooks*** в **Excel** є аналогом сімейства ***Documents*** в **Word** і містить набір об'єктів ***Workbook*** (аналог об'єкта ***Document*** в **Word**), кожен з яких відповідає відкритій в **Word** робочій книзі. Нова робоча книга створюється за допомогою методу ***Add*** об'єкта ***Workbooks***.

Для доступу до осередків активного робочого листа **Excel** використовується властивість ***Cells*** об'єкта ***Application***. У загальному випадку, вона повертає об'єкт ***Range***, який представляє набір комірок. Для отримання або зміни значення окремої комірки застосовується конструкція ***Cells.Item(row,column).Value***, де ***row*** і ***column*** є відповідно номерами рядка і стовпця (починаючи з одиниці), на перетині яких вона знаходиться.

Перепишемо приклади сценаріїв з документами **Word** та **Excel** з ЛР № 5 на мові **PowerShell**.

Файл toWord.ps1

```
$d=Get-Date      # отримуємо поточну дату
$strText=$d.Day.ToString()+»/»   # витягуємо день
$strText+=$d.Month.ToString()+»/»  # місяць
$strText+=$d.Year.ToString()+»`n»  # рік
$strText+=«Доданий рядок зі сценарію PowerShell`n»
```

```
$oWord=New-Object -ComObject Word.Application  # створюємо новий об'єкт Word
$oDoc=$oWord.Documents.Add()   # створюємо новий документ
$oWord.Visible=$true      # робимо вікно Word видимим
$oDoc.Content.InsertAfter($strText)
$oDoc.SaveAs(«D:\From PowerShell.doc»)
$oDoc.Close()
$oWord.Quit()
```

Файл toExcel.ps1:

```
# створюємо об'єкт - excel-додаток
$objXL=New-Object -ComObject Excel.Application

# робимо вікно видимим і створюємо робочу книгу
$objXL.Visible=$true
$objXL.WorkBooks.Add()

# встановлюємо ширину першого стовпця
$objXL.Columns.Item(1).ColumnWidth=20;

# записуємо рядок у комірку (1,1)
$objXL.Cells.Item(1,1).Value=«Створено зі сценарію PowerShell»
```

Робота з .NET-об'єктами на прикладі Windows.Forms

Платформа .NET побудована таким чином, що для звернення до тих чи інших об'єктів потрібно попередньо завантажити в операційну пам'ять відповідну збірку (assembly) – динамічну бібліотеку певного виду. Найчастіше використовуються збірки, які завантажуються в **PowerShell** автоматично, їх список можна побачити за допомогою статичного методу ***getAssemblies()*** наступним чином:

PS C:\> [AppDomain]::CurrentDomain.GetAssemblies()

Наприклад, для звернення до об'єктів ***WinForms*** потрібно за допомогою методу ***LoadWithPartialName*** завантажити збірку, що підтримує об'єкти (результат виконання методу приводиться до типу ***void*** для придушення виведення на екран непотрібної інформації):

PS C:\> [void][System.Reflection.Assembly]:: LoadWithPartialName («System.Windows.Forms»)

Наведемо приклад сценарію PowerShell, що відкриває форму, на якій розміщені 2 кнопки (**«OK»** та **«Cancel»**), поле для введення тексту та напис для виведення тексту:

CreateForm.ps1

```
#створення нової форми, встановлення її заголовку, ширини та висоти
$form=New-Object System.Windows.Forms.Form
$form.Text=«Перша форма»
$form.Width=200
$form.Height=200

# створення кнопки «OK»
$button1=New-Object System.Windows.Forms.Button
$button1.Text=«OK»
$button1.Top=20
$button1.Left=20
$button1.Width=75
$button1.Height=23

# створення кнопки «Cancel»
$button2=New-Object System.Windows.Forms.Button
$button2.Text=«Cancel»
$button2.Top=45
$button2.Left=20
$button2.Width=75
$button2.Height=23

# розміщення кнопок на формі
$form.Controls.Add($button1)
$form.Controls.Add($button2)

# створення текстового поля для введення
$textbox=New-Object System.Windows.Forms.TextBox
$textbox.Top=75
$textbox.Left=20
$form.Controls.Add($textbox)

# створення напису для відображення тексту
$label=New-Object System.Windows.Forms.Label
$label.Top=110
$label.Left=30
$label.Text=«Label: «
$form.Controls.Add($label)
```

Визначимо тепер дію, яка виконуватиметься під час натискання кнопок. Для цього потрібно написати обробник події **Click** кнопки (тобто вказати, які команди повинні виконуватися під час натиснення на кнопку). Оброблювач подій – це спеціальний метод з назвою ***Add_подія***.

```
# додавання оброблювачів подій натиснення на кнопки
# кнопка «ОК» дублює на написі введений текст
# кнопка «Cancel» закриває форму
$button1.Add_Click(
{
   $text=$textbox.Text
   $label.Text=$text
}
)

$button2.Add_Click(
{
   $form.Close()
}
)

# запускаємо форму
$form.ShowDialog()
```

Завдання

1. Написати сценарій **PowerShell**, який аналогічно до скрипту WSH з ЛР № 5 приймає введені від користувача рядки, створює нову книгу Excel, документ Word та записує у них введені користувачем рядки (комірки, у які будуть записуватися дані у Excel обрати на власний розсуд). Під час введення користувачем слова «Quit» завершити введення даних та зберегти обидва файли.
2. Написати сценарій **PowerShell**, що запускає форму, на якій розміщені 3 кнопки та поле для введення. Кнопки мають написи «Приховати», «Показати» та «Очистити». Відповідні обробники подій будуть приховувати, показувати поле введення та видаляти у ньому текст.

Контрольні питання

1. Як створити новий об’єкт у **PowerShell?** Якими можуть бути ці об’єкти?
2. З чого складається об’єктна модель документів Word та Excel? Як відкрити нові документи Word та Excel у сценаріях **PowerShell?**
3. Які дії потрібно зробити, щоб стало можливим запустити форму зі сценарію **PowerShell?**
4. Як у сценарії **PowerShell** додати візуальні елементи до форми та створити їх обробники подій?

Навчальне видання

Гліб Валентинович
Горбань

МЕТОДИ ПРОГРАМУВАННЯ ПІД ОПЕРАЦІЙНІ СИСТЕМИ

Методичні вказівки

Випуск 365

Редактор *Р. Грубкіна.*
Технічний редактор *О. Петроченко.*
Комп'ютерна верстка *Д. Кардаш.*
Друк *С. Волинець*, фальцювально-палітурні роботи *О. Мішалкіна.*

Підп. до друку 07.09.2021
Формат 60x84$^{1}/_{16}$. Папір офсет.
Гарнітура «Times New Roman». Друк ризограф.

Ум. друк. арк. 5,34. Обл.-вид. арк. 3,60.
Тираж 5 пр. Зам. № 6318.

Видавець і виготовлювач: ЧНУ ім. Петра Могили.
54003, м. Миколаїв, вул. 68 Десантників, 10.
Тел.: 8 (0512) 50–03–32, 8 (0512) 76–55–81, e-mail: rector@chmnu.edu.ua.
Свідоцтво суб'єкта видавничої справи ДК № 6124 від 05.04.2018.